Монитор

журнал для программистов — про

PROGRAMMER

Языки программирования четвертого поколения

Шифрование: Эль Гамаль и электронная подпись

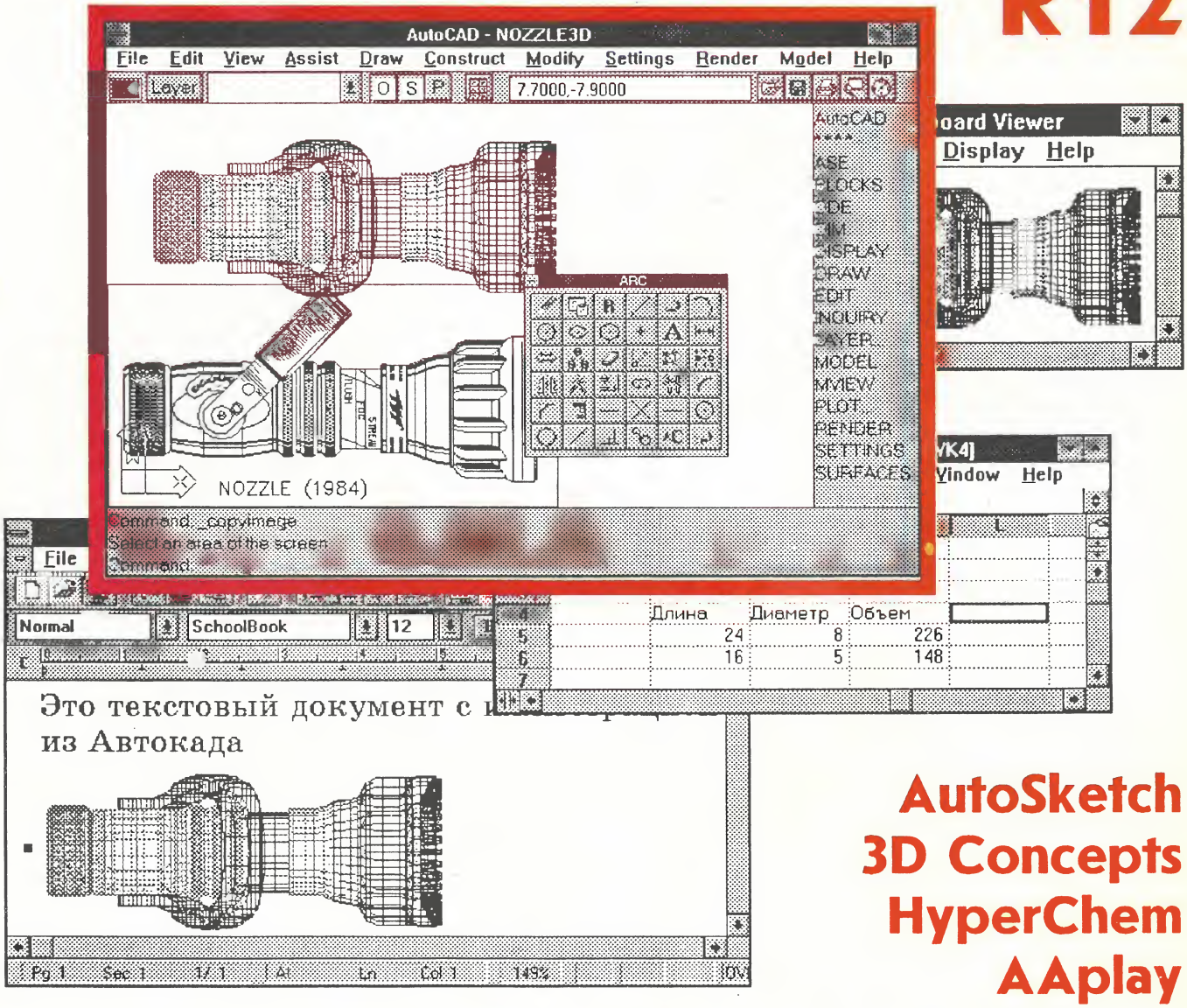
Работа со звуком

ESSAYS

Отечественное программирование

Autodesk®

AutoCAD® R12



AutoSketch
3D Concepts
HyperChem
AAplay

for Windows™

на выставке Windows EXPO™ / Moscow'93
с 29 ноября по 2 декабря 1993 года
ВВЦ (бывш. ВДНХ), павильон 4

Старший брат смотрит на тебя
Джордж Оруэлл. «1984»

В антиутопии Оруэлла описано общество, организовавшее тотальный контроль за поведением, образом жизни и привычками отдельных людей. Оруэлл завершил свою книгу в 1948 году, когда ученые Соединенных Штатов и Советского Союза уже вели разработку первых цифровых компьютеров. Сегодня развитие компьютерных технологий и повсеместное внедрение информационных систем привели к тому, что контроль за частной жизнью не только стал технологически реальным, но и осуществляется в той или иной степени во всех развитых странах.

Практически каждое действие гражданина США, Западной Европы, Японии, связанное с общением с государственными органами или частными фирмами, фиксируется в компьютерных системах. Информация о покупках, оплаченных с помощью кредитной карточки, попадает как в компьютер магазина, так и в компьютер банка, выдавшего Вам кредитную карточку. Все поступления на Ваш расчетный счет и Ваши траты также запоминаются в компьютере банка. Информационная система поликлиники, где Вы лечитесь, содержит полную информацию о Вашей истории болезни и состоянии здоровья. В компьютере телефонной станции фиксируются все Ваши телефонные переговоры. Компьютеры телевизионных каналов, абонентами которых Вы являетесь, следят за тем, какие передачи на протяжении какого времени Вы смотрите. Налоговая служба хранит в компьютере информацию о Ваших доходах; транспортная полиция — о допущенных Вами нарушениях правил дорожного движения; Ваш работодатель — о Вашей служебной деятельности.

Современные системы контроля доступа в служебные помещения, дополненные скрытым телевизионным наблюдением и прослушиванием телефонных переговоров, позволяют контролировать местонахождение каждого сотрудника и гостя фирмы, лояльность сотрудников и их стопроцентную занятость в рабочее время делами фирмы.

Все это делает возможным сбор практически полной информации о конкретном человеке путем договоренности с держателями информации, подкупа или технического шпионажа. Возможность эта не раз эффектно демонстрировалась: например, в 1989 году американский репортер, подкупив банковского чиновника, приобрел и опубликовал сведения о покупках, сделанных с помощью кредитной карточки тогдашним вице-президентом США Дэном Куэйлом.

Такая опасность реальна не только для высокопоставленных политиков и бизнесменов, находящихся под постоянным наблюдением журналистов и конкурентов. Существует множество организаций, заинтересованных в постоянном контроле за жизнью рядовых людей. Налоговая инспекция стремится получить полную информацию о Ваших доходах, милиция и службы безопасности рассматривают каждого человека как потенциального преступника, жертву преступления или свидетеля. Если Вы — иностранец и германская полиция задержала Вас при управлении машиной в нетрезвом состоянии, то информация об этом попадет в компьютеры всех германских посольств и, возможно, въезд в Германию будет для Вас навсегда закрыт. Страховая компания, заполучив Вашу историю болезни, может резко увеличить размер страхового взноса, а банк — отказать Вам в кредите. Работодатели при приеме на работу собирают информацию о своих будущих сотрудниках; фирмы-производители — о вкусах и привычках своих покупателей.

Вхождение в нашу жизнь компьютеров, кредитных карточек, страховых компаний и прочих атрибутов западной цивилизации в сочетании с унаследованными от недавнего прошлого нравами и неразвитость ограждающих права человека законов делают проблему противостояния отдельного человека контролю за своей частной жизнью весьма актуальной для России.

Существуют разные способы защиты от злоупотребления собранной о человеке информацией.

Во-первых, должен быть принят закон, по которому каждый гражданин имеет право познакомиться с информацией, собранной о нем любым государственным учреждением или частной фирмой.

Во-вторых, должна быть запрещена передача информации, собранной государственными учреждениями или частными фирмами, кому бы то ни было, за исключением четко оговоренных случаев, связанных с расследованием преступлений и поддержанием общественной безопасности.

Третий и, может быть, самый важный способ защиты личности — развитие анонимных технологий. Это означает, что любая информация о конкретном человеке должна храниться в информационной системе лишь до тех пор, пока она необходима. Например, информация о личности нарушителя правил движения хранится в компьютере ГАИ лишь до момента уплаты штрафа. Информация о телефонных разговорах уничтожается компьютером телефонной станции, как только счет за разговоры оплачен, а информация о личности покупателя стирается в компьютере магазина, как только счет за покупку оплачен выдавшим кредитную карточку банком. Сами кредитные карточки могут приобретаться за наличные и не содержать информации о личности владельца.

Эти строки пишутся в надежде на то, что прочитавшие их проектировщики информационных систем задумаются, не могут ли результаты их труда быть использованы для чрезмерного контроля за частной жизнью людей.

Журнал для программистов-практиков

Редколлегия:
Д.Э.Бройтман
(главный редактор)
В.В.Водолазский
В.С.Головин
Н.Н.Косарев
А.М.Миляков
(ответственный секретарь)
Н.А.Никитина
(отдел рекламы)
Е.В.Равич
С.М.Тимашова
Л.И.Турок

Графическое бюро:
С.В.Макельский
(главный художник)
Г.Л.Карчикян
(компьютерная верстка)

Отдел распространения:
Р.А.Болдырев
А.А.Винокуров
Е.В.Гун

Дирекция
А.В.Орехов
(издательский директор)
А.Д.Гассельблат
(коммерческий директор)

Представительство в Европе (Германия)
Marina Guetmanskaia
GLEIWITZER STR 5
67659 KAISERSLAUTERN
Tel. (0631) 7-49-05

В оформлении номера использованы рисунки
А.С.Замулы
Д.Н.Калининой

Подписано в печать с оригинал-макета 27.09.93
Формат 60x84/8.
Тираж 30 000 экз.
Печать офсетная. Печ. л. 10,0
Цена договорная
Зак. 4695
Отпечатано в типографии ИПК «Московская правда»

Учредитель:
Научно-техническое общество

Софт·Москва

Для писем: 117419 Москва, а/я 765
Тел. (095) 238-16-22
Факс (095) 238-32-78
Для справок (095) 943-25-53
Телекс 613322 GRANT SU
BBS (095) 135-14-39 ARSORBIS

© "Монитор", 1993

Письма

Turbo Vision: вопросы и ответы. Как приобрести документацию к игровым программам. Судьба книги «Функции Turbo C, C++» — 4

Эссе

А. Анненков. Отечественное программирование — 8

Языки программирования

А. Штрик. Языки программирования четвертого поколения. Часть 1 — 10

Рассказывается о причинах, приведших к появлению языков 4GL; описываются основные черты языков программирования этого класса.

М. Трофимов. Специфические ошибки Turbo Pascal — 16

Программируя на Turbo Pascal, легко сделать ряд трудно обнаруживаемых ошибок, в то время как более старые компиляторы позволяют их избежать. В статье рассмотрен ряд типичных ошибок и обсуждаются некоторые методы борьбы с ними.

MS BASIC

А. Колесов. Профессиональная работа в среде Microsoft BASIC. Часть 3 — 22

Рассматриваются проблемы, возникающие при создании больших программ, даются практические рекомендации по минимизации размеров исполняемых модулей и сокращению времени компиляции.

Технологии программирования

С. Никоненко. Программируем СУБД на Turbo Vision. Часть 3 — 28

Продолжение цикла статей о библиотеке Data Vision. Обсуждается методика создания виртуальных баз данных, объединяющих файлы различной физической структуры, а также способ включения в состав базы фиктивных полей.

Защита программ

В. Водолазкий. Коммерческие системы шифрования: основные алгоритмы и их реализация. Часть 3 — 34

Статья завершает цикл, посвященный коммерческим системам шифрования. Описываются алгоритм «упаковки рюкзака» и система ElGamal. Обсуждается принцип реализации цифровой подписи, основанный на системах с открытым ключом.

Мир графики

Е. Кисель. **Эффекты для демонстрации слайдов** **40**

В общем виде постановку задачи можно сформулировать так: мы хотим научиться писать программы, отображающие на экране последовательности слайдов с использованием эффектов.

С. Курковский. **Стандарт PCX + эффекты** **48**

В статье приводится библиотека функций, позволяющих создавать различные эффекты при выводе на экран изображений в формате PCX.

DOS под микроскопом

Д. Валдин. **Резидентные программы на языке С. Часть 2** **52**

Продолжение статьи о резидентных программах на языке С. Обсуждаются вопросы предотвращения повторной загрузки в память и выгрузки программы из ОЗУ.

Компьютер в разрезе

Д. Солдатенков. **Звуковые устройства** **62**

Статья посвящена путям расширения звуковых возможностей IBM PC. Приведен исходный текст программы на Turbo Pascal, демонстрирующей возможности встроенного динамика.

Д. Правиков. **Измерение длины дорожки дискеты** **66**

Статья содержит описание алгоритма оценки длины дорожки, обеспечивающего максимально достижимую точность измерений.

Банковские системы

Копейка рубль бережет, или о том, как денежки счет любят **69**

Сети

Е. Морозов, Е. Федорова. **Средства сетевых СУБД для восстановления баз данных** **70**

На примерах конкретных сетевых СУБД рассматриваются средства восстановления баз данных после отказов.

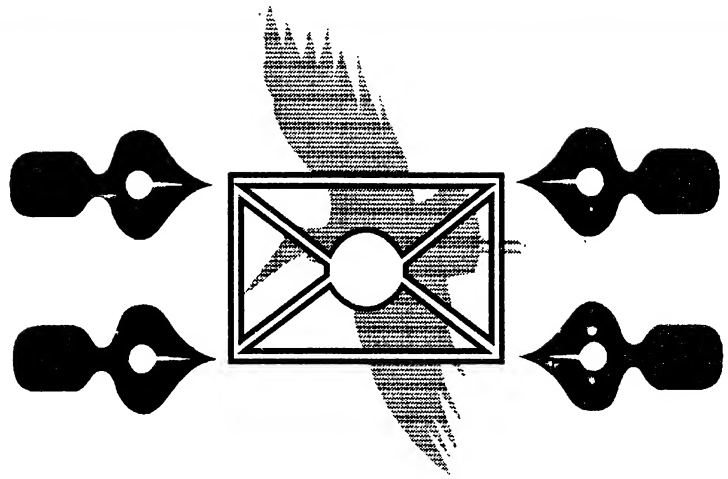
Лицей

А. Шибенко. **Резидентный «будильник» на Turbo Pascal 5.0** **74**

Игры

В. Водолазкий. **Первым делом— вертолеты...** **76**

Впервые в нашем журнале публикуется статья про авиационный имитатор — игру Comanche Maximum Overkill фирмы NovaLogic, имитирующую управление легким тактическим вертолетом огневой поддержки RAH-66 Comanche.



Здравствуйте, уважаемая редакция!

Пишет вам один из начинающих «программистов-практиков» (МЭСИ 1-й курс, прикладная математика). И как у любого начинающего, у меня много вопросов, хотя на некоторые из них ответы уже появляются.

Чтобы не казаться голословным, начну сразу с ответа (если сочтете нужным, то это можно поместить в виде маленькой заметки).

Каждый, кто начинает программировать на Turbo Vision, в первой же своей программе (а скорее всего, после того как нажмет Enter в About диалоге интегрированной среды Turbo Pascal) сталкивается с ситуацией, когда на панели экрана (DeskTop) ничего нет, а для того чтобы попасть в меню, надо обязательно нажать F10 или Alt+<«горячая» клавиша>.

Далее приведен листинг метода Idle.Application или его потомка с одним из возможных вариантов решения этой задачи.

```
Procedure TApplication.Idle;
Var
  E : TEvent;
begin
  IF Application^.Current = DeskTop THEN
  { в группе Application активна DeskTop }
     IF DeskTop^.Current = nil THEN
        begin
           E.What := evCommand;
           E.Command := cmMenu;
           PutEvent(E);
        end;
  TProgram.Idle;
end;
```

Теперь вопросы, ответы на которые, я думаю, интересуют не только меня.

1. Русификация Turbo Vision («горячие» клавиши в меню, кнопках и так далее).
2. В среде TP 6.0 строка статуса выглядит при сохранении файла следующим образом:

```
saving <имя файла>
```

Интерес представляет именно saving <имя файла>, а не просто saving...

3. В среде TP 6.0 в меню Compile/Destination опция Disk/Memory каким-то образом меняется. Как бы это сделать?

Конечно, у меня есть идеи ответов на вопросы (1 — GetAltCode, 2 и 3 — найти указатель на нужную строку. Но идеи ответов на Pascal не напишешь, и поэтому мне бы очень хотелось, чтобы Вы не оставили эти маленькие вопросы на страницах вашего журнала без ответа.

И, честно говоря, вообще было бы неплохо, если бы в «Мониторе» была рубрика типа «Вопросы => Ответы».

С. Киселев

Начнем с Вашего ответа на Вами же заданный вопрос. Предложенная схема вполне работоспособна и используется многими программистами. В частности, она реализована в демонстрационной версии пакета Data Vision, к которому мы еще вернемся. И, кстати, очень правильно, что для генерации события Вы не используете функцию Message. В контексте решаемой задачи это просто лишнее. Есть лишь один недостаток. Опрос метода Idle, как Вам, вероятно, известно, осуществляется системой постоянно в цикле ожидания ввода от пользователя. Если Вы не включаете в состав метода никаких дополнительных функций, кроме описанной Вами (ну, разве что добавите индикатор свободной памяти и часы), то такое решение вполне приемлемо. А если программа использует фоновую обработку?

По-моему, гораздо эффективнее, с точки зрения затрачиваемого программой времени, расширить предлагаемым Вами фрагментом метод ClearEvent объекта TView, который наследуется без изменений вплоть до TApplication (в чем можно убедиться, просмотрев исходные тексты Turbo Vision).

```
procedure TView.ClearEvent(var Event: TEvent);
begin
  Event.What := evNothing;
  Event.InfoPtr := @Self;
end;
```

Как видите, по умолчанию ClearEvent очищает все поля события. Ну, так подставьте вместо них cmMenu, и, поскольку метод **не виртуальный,** *генерация события будет выполняться только для обработчика событий в Вашем потомке от TApplication.*

Вопрос о русификации Turbo Vision также имеет свое решение. Путей много, но самый эффективный — тотальная переделка всего ядра. Работа это кропотливая, но, к счастью, уже проделанная несколькими коллективами умельцев. Я лично пользуюсь системой Data Vision разработки фирмы SHIFT из Тамбова. О ней, кстати, мы публикуем ряд статей, начиная с «Монитора» 3.92, и от желающих приобрести нет отбоя (не в обиду нежелающим будь сказано).

С сохранением нового файла ситуация еще проще, чем Вам кажется. В меню **Files** *есть пункт* **Save as...**, *при выборе которого Вас спросят об имени файла. Почему бы этот пункт не выполнять при нажатии на «горячую» клавишу? По всей видимости, из-за того, что это не столь уж часто используемая операция. А необходимость нажимать несколько клавиш подряд заставит Вас тщательно продумать свои шаги, и, следовательно, шансов случайно стереть уже существующий файл в результате перезаписи у Вас меньше.*

Что касается изменения строк в меню, то эта проблема интересует многих, но удовлетворительного решения на глаза мне, например, не попадалось. Один из вариантов, который был опубликован в PC Magazine, — это уничтожение экземпляра объекта TMenuBar и генерация нового с другим NewSItem в соответствующей строке. Кардинальное же исследование этой проблемы, если таковое пришлет кто-нибудь из читателей, будет напечатано в нашем журнале.

И, наконец, что касается рубрики «Вопросы => Ответы». Как пользователь Pascal Вы согласитесь со мной: из Вашей формулировки вытекает, что Вопросов **не** **мельше**, *чем ответов. Ограниченных сил нашей редакции не хватает на поток писем, заваливающих наш а/я 765. Поэтому часть вопросов мы будем публиковать без ответов в надежде, что читатели, владеющие необходимой информацией, поделятся ею через наш журнал. Ниже мы приводим такой «крик души» юных фанатов нашего журнала.*

В. Водолазкий ■

Здравствуйте, уважаемые господа!

Пишет Вам это письмо Ваш постоянный читатель с такой просьбой.

Мы с моим другом не знаем, где можно приобрести документацию к игровым программам:

STARBLADE (SILMARILS)
COLORADO (SILMARILS)
TARGHAN (SILMARILS)
METAL MUTANT (SILMARILS)

Мы думаем, может быть, Вы нам поможете. Главным образом, нас интересуют коды. Вот например, в **Colorado** появляется такая надпись, и мы не знаем что делать.

LA VISAGE PALE A-T-1L
LA LANGUE FOURCHUE?
OUVRE LE MANUEL ET
ECRIS LE NOT SUIVANT
LA PAGE: 4
LA SIGNE: 4
MOT NUMEкO: 4
?????????

С нетерпением жду Вашего ответа!

С. Романов

Уважаемый господин Романов.

Мы решили познакомить с Вашими проблемами всех читателей нашего журнала и надеемся, что они Вам помогут. Более того, у нас уже есть предложение от одного из читателей насчет статьи об игре **Metal Mutant.** *Может быть, появятся публикации и по другим играм фирмы* **SILMARILS.** *К сожалению, буржуазные производители компьютерных игр* **не желают** *продавать свои игры в России. Поэтому либо мы будем писать для себя игры сами, а такие прецеденты уже есть (см. «Монитор-Аспект» 2.93), либо будем заниматься «пиратством» по отношению к разноцветным миазмам империалистической компьютерной мысли. И никакие высокоморальные заявления положения дел не поправят! А стало быть, у Вас все-таки есть шанс получить ответы на свои вопросы.*

В. Водолазкий ■

Здравствуйте!

Прочитал в журнале «Монитор» ваше объявление о продаже наложенным платежом книги «Функции Turbo C, C++». Меня очень интересуют книги и документация по программированию на языках Turbo C, C++, Clipper 5.01. Прошу Вас выслать каталог продаваемых (высылаемых) Вами книг.

В. Коробов

Это далеко не единственное письмо, в котором нас просят выслать книгу по С. Увы, книга так и не вышла из печати, и мы вынуждены извиниться перед нашими читателями за ложную тревогу. Но зато мы можем предложить нашим читателям следующие (пока только две) книги.

1. Гради Буч. Методология объектно-ориентированного программирования с примерами.– К.: Диалектика. Цена 2800 руб. (50 экз.)

2. Справочное руководство по Object Windows для C++ в двух частях.– К.: Диалектика. Цена 1500 руб. (50 экз.)

Цены приведены с учетом пересылки. В скобках указано количество книг на складе.

А. Винокуров ■

Здравствуйте, уважаемая редакция журнала «Монитор».

Пишет Вам программист, разрабатывавший в свое время компиляторы для сильно типизированных и объектно-ориентированных языков программирования, в частности, Modula-2, Oberon, Object Oberon. Опубликованная в Вашем журнале статья М. Трофимова [1] содержит, на мой взгляд, несколько ошибок.

Фраза «Дело в том, что в современных компиляторах механизмы рекурсивного вызова реализуются настолько эффективно, что...» неверна, так как эти механизмы там вообще никак не реализуются. Компилятор вообще не в состоянии распознать рекурсивный вызов, поскольку он может быть как косвенным (процедура вызывает саму себя через вызов других процедур), так и динамическим — через процедурную переменную.

Строго говоря, рекурсивный вызов — не элемент синтаксиса и даже семантики языка, и определение языка его не содержит. Это просто один из способов описания алгоритма. Компилятор не следит за тем, откуда вызывается процедура. Элементом семантики языка является рекурсивная декларация типа, за чем компилятор действительно следит.

Корректность рекурсии обеспечивается таким способом реализации, при котором резервирование самой процедурой на стеке области для локальных переменных и передача параметров через стек создают при активизации любой процедуры свою уникальную область ей принадлежащих переменных (такая область обычно называется кадром, или фреймом).

Генерация кода процедуры делается контекстно-независимо, с одним маленьким исключением в случае, если это процедура вложенная (Pascal, Modula-2 и другие) и внутри нее есть обращения к переменным охватывающей процедуры (такой доступ оформляется в виде специального кода). Процедура внутри себя не распознает, кто и как ее вызвал. Кстати, упоминаемый запрет на рекурсию в таких языках, как Fortran и Basic, вытекает из того простого факта, что там параметры и переменные всегда глобальны, поэтому вызывающая сама себя процедура будет работать с одними и теми же экземплярами переменных, что и приведет к некорректной работе.

Таким образом, необходимым (но не достаточным) условием возможности рекурсии в языке является обеспечение для процедуры собственной области локальных переменных и параметров каким-либо образом (для этого может использоваться и динамически распределяемая область памяти). Если же процедура модифицирует свой код, то этого условия недостаточно и рекурсия работать не будет. Отсюда часть советов в последнем абзаце на стр. 54 некорректна. Размер кода упоминаемой процедуры Add совершенно неважен для эффективности рекурсии. **Код рекурсивной процедуры не размножается динамически. Код всегда существует в одном экземпляре, и этого вполне достаточно.** Если автор статьи в состоянии предложить пригодный для компилятора способ распознавания рекурсивного вызова, то ему нужно готовить статью в толстый англоязычный журнал.

Совет же о сокращении одних и тех же вычислений справедлив, поскольку эквивалентен совету о чистке цикла. Но это не особенность рекурсии.

Исходя из изложенного, могу сформулировать «Советы старого программиста о рекурсии».

Рассмотрим процедуру P(fp : INTEGER);

Эффективность рекурсии по времени определяется накладными расходами по:

передаче параметров в стек	PUSH fp
вызову процедуры (например, статически)	CALL P
командам пролога процедуры	ENTER длина области локальных переменных;
.....	
возврату из процедуры	LEAVE RET 2

Поверьте, что в тактах это немного.

Итак, советы таковы:

- рекурсию используйте смело;
- сокращайте размер фрейма процедуры (параметры и локальные переменные), если переполняется стек;
- помните, что процедура, вовлеченная в рекурсию, должна иметь свою копию параметров (если они модифицируются). Если параметры — параметры-значения неструктурных типов, то это происходит автоматически, для параметров-переменных это не так, и процедура P(var fp : integer) может и не работать в рекурсии. Более тонкий случай — параметры-значения структурных и длинных типов. Классически передача таких параметров реализуется так: при вызове в стек передается адрес параметра (как будто этот параметр — переменная), а в прологе процедуры создается код для копирования этого параметра в стек с использованием указанного адреса, после чего адрес корректируется. Такое копирование и корректировка адреса обычно подавляется опциями компилятора (для эффективности), и если копирования не происходит, то модификация такого параметра в процедуре, скорее всего, приведет к неверной работе;
- рекурсивный вызов всегда должен происходить по определенному условию;
- безусловный вызов всегда некорректен, поскольку просто отсутствует условие останова рекурсии и программа переполнит стек.

Некоторый терминологический материал, полезный для пытливого читателя

Упоминаемый в статье [1] метод построения компиляторов называется «top-down analysis», то есть «разбор сверху вниз» или «рекурсивный спуск».

Метод очень простой и мощный для определенного класса грамматик, к которым относятся и грамматики языков Pascal, Modula, Oberon, ADA, VHDL и другие. Скажем, грамматики таких языков делались специально под такой метод разбора.

У этого метода существует один недостаток, распространяющийся на рекурсивные алгоритмы вообще, — «рекурсию трудно остановить».

Так, забравшийся при анализе программы в глубокую рекурсию и обнаруживший ошибку компилятор должен корректно «выбраться» из рекурсии и передать сообщение редактору текста. Сделать это можно либо продолжая компиляцию до места, пока мы из рекурсии не выберемся — до конца компилируемого

файла (что, вообще говоря, невозможно, например, если конец текста программы отсутствует), либо реализовав возврат в каждой процедуре, вовлеченной в рекурсию, что приведет к сильному усложнению алгоритма, либо методом восстановления контекста (что существенно проще). Для такой операции часто в системах программирования существуют процедуры типа «GetContext» и «SetContext». Первая просто списывает все регистры в указанную область памяти, а вторая их восстанавливает. Таким образом, взяв контекст перед входом в рекурсию, а затем для экстренного выхода восстановив его, мы окажемся в ситуации, как будто мы в рекурсию и не входили. Этот способ не вполне корректен, так как состояние регистров не является полным контекстом программы и автору этих строк приходилось разбираться в оверлейном механизме Pascal 6.0 для перепрыгиваний в выгруженный оверлей, но для большинства случаев он применим.

Пытливый читатель, возможно, удивится приведенному в статье [1] факту, что компилятор языка часто пишется на самом языке, и подумает, что это тоже какая-то рекурсия. Действительно, зачем писать компилятор, если он уже есть? Этот метод называется «раскруткой» и вначале берется либо компилятор с близкого языка, например, для Modula — уже существующий Pascal, либо для «голой», но с ассемблером машины — ассемблер. Затем методом последовательного приближения пишутся все более и более полные подмножества конечного компилятора на предыдущей его версии. Кстати, такой тест, когда компилятор компилирует сам себя и результат компиляции остается компилятором, а не зависающей программой, является мощным тестом на корректность реализации языка (красота рекурсии в чистом виде!!!). Время же такой компиляции является комплексным тестом эффективности реализации языка (качества кода, скорости работы алгоритма компиляции и так далее). Такой тест вообще дает массу пищи для философских рассуждений о рекурсии, поскольку компиляторы — единственные из программ, способные «порождать» сами себя.

Упоминаемая в статье [1] машина Lilith весьма интересна. Этот 16-битовый процессор и машина целиком были созданы командой Вирта, кажется, из трех человек. Предпосылкой для создания была идея процессора с системой команд, ориентированной на язык высокого уровня, изначально Modula-2, и аппаратную поддержку динамической загрузки модулей. Под динамической загрузкой понимается такой способ функционирования программ, при котором в системе существуют только отдельные модули, но нет механизма связывания модулей в исполняемые файлы, то есть две одновременно запущенные задачи пользуются одним и тем же кодом какого-либо модуля. Модули по мере обращения к ним подгружаются в память, а неиспользуемые выгружаются. Таким образом, не существует копий кодов модулей в исполняемых файлах, экономится место и нет необходимости повторного редактирования при изменении неинтерфейсной части текста модуля.

В этом стековом процессоре система команд подобрана так, что все управляющие операторы Modula-2 реализуются соответствующей машинной инструкцией. Специальные инструкции применяются для вычисления булевых выражений (системы команд классических микропроцессоров слабо поддерживают такие вычисления, что приводит к усложнению компиляторов и кода). Для обращения к процедуре используется метод динамического вызова через номер процедуры и номер модуля. Генератор кода компилятора Modula-2 для такой архитектуры получается раз в 5 короче, чем, например, для Intel, а сам код примерно в 2 раза более компактен. Соответственно повышается и производительность процессора. В Новосибирске в свое время было создано 32-битовое расширение этой архитектуры под названием «Кронос».

Вообще система эта очень специфичная и приводить пример в пользу рекурсии, исходя из тестов, не очень корректно.

Такой подход к разработке процессоров (Н. Вирт) является вполне логичным, так как 99% программ написаны на языках высокого уровня и скорость работы процессоров не следует измерять в регистр-регистровых операциях, а следует позаботиться об адекватности системы команд процессора основным конструкциям современных языков программирования, что, впрочем, не мешает Intel выпускать приборы с командами двоично-десятичной арифметики, хотя эту совместимость с семисегментными индикаторами можно было бы и не поддерживать.

С уважением,

А. Лысогоров ■

Литература

1. М. Трофимов. Итерация против рекурсии: проблема выбора// Монитор.– 1992.– № 8.

Анонс

«Монитор» — версия 7.93

А. Морозова. Отечественные CASE средства разработки проектов информационных систем
Даются основы CASE подхода как нового направления программной инженерии. Оценивается состояние отечественной CASE технологии в области автоматизации разработки проектов информационных систем. Приводится краткое описание десяти CASE средств, созданных в России в последние годы.

А. Тихонов. TopSpeed Pascal
В статье рассмотрена система программирования TopSpeed версии 3.1 фирмы Clarion Software, обладающая рядом уникальных особенностей. Анализируются особенности входного языка TopSpeed Pascal, совместимость его с Turbo Pascal, языками семейства TopSpeed и другими системами программирования.

С. Курковский. Интервальные методы в компьютерной графике
В статье рассматриваются оригинальные алгоритмы для построения графиков, линий уровня и поверхностей различных функций. Приведенные алгоритмы основаны на применении интервальной арифметики, которой посвящен один из разделов статьи. Большое внимание уделяется практической реализации описанных методов на языке C++.

Д. Мастрюков. Алгоритмы сжатия информации. Часть 1. Сжатие по Хаффмену
Первая статья из цикла статей, посвященных алгоритмам сжатия информации. В первой части рассматривается алгоритм адаптивного сжатия по Хаффмену.

О. Шарапов. Загружен — не загружен, загружен — не загружен, загружен...?
В статье рассматривается проблема определения повторной загрузки резидентной программы в память компьютера.

Отечественное программирование

А. Анненков

Вначале было программирование. Чуть погодя оно стало модульным, затем структурным. Теперь — объектно-ориентированным. И кто знает, что наступит завтра.

Будем, однако, помнить о главном. Технологии приходят и уходят, при этом само программирование никуда не девается. «Танки, господа, это мода, а кавалерия вечна». Как и на чем бы мы ни писали, наш труд был, есть и будет программированием. Позвольте конкретизировать — отечественным программированием (ОП).

ОП — это феномен, заслуживающий пристального внимания. Действительно, каким образом наша страна, где программист не в состоянии приватизировать компьютер, держит паритет с кем хочешь в области прикладного программирования? Это явление сопоставимо с созданием ядерного щита или, скажем, с тем фактом, что Китай кормит себя сам. Впрочем, и нас, и китайцев можно понять. Они хотят кушать, а мы, настроенные менее прозаично, тянемся к возвышенному.

Немного истории. Ведь то, что можно наблюдать сегодня, не всегда было таким и вообще не всегда было. Старики рассказывают, что в машинных залах БЭСМ-6 они не слышали английского. Не из-за шума, нет — за ненадобностью. В те былинные времена наиболее английским было слово «фортран-Дубна». И эта БЭСМ-6, грохоча перфовводом, обеспечивала своим счастливым, уверенным в завтрашнем дне пользователям миллион операций в секунду, виртуальную память и разделение времени. Да что БЭСМ... Спросите нынешних программистов, что такое Бармалей. Ответит ли кто-нибудь?

Ответят те, кто еще способен отличить ЕС ЭВМ от интерьера котельной. Эти ЕС ЭВМ! Их грязно-голубые шкафы стали символом национального унижения. Удостоенные государственной премии пуды документации ОС ЕС, представляющие собой грубый подстрочник американских текстов... Еще живы те, кто их читал. Некоторые, возможно, помнят фамилию главного лауреата и даже знают, что Шура-Бура и Бармалей — не одно и то же.

Тем не менее, ОП выстояло в пятилетки лихолетья АСУ. На сбоившей каждые полчаса технике выполнялись нужные программы (автору доподлинно известны случаи машинного расчета заработной платы), и это не оставляет сомнений в том, что продукты ОП — самые надежные в мире. Попробуйте-ка жаркой летней ночью при выключенных кондиционерах отладить на ЕС-1022 что-либо длиннее двадцати операторов PL/1! Бедного программиста эта изумительная техника моментально брала на джоб-стоп. И все же ночь, проведенная с машиной, была счастьем. Машинное время, подобно Граалю, являлось предметом вожделения и безумных желаний, которые никому не суждено утолить до конца.

Для ОП это была полноценная, бурная жизнь, кипевшая шекспировскими страстями. Системные программисты пускали в ход кулаки, когда нарушались их идиотские инструкции по назначению приоритетов, а пользователи в отместку разрешали своим заданиям сворачивать чужую память. Программист, не стыдясь, интриговал или просто заискивал перед девушкой-оператором, дабы та поскорее принесла распечатку или сменила диск. Начальника ВЦ (не путать с WC) в глубине души побаивались. Словом, в стране был порядок.

Найдутся юные нахалы, полагающие, что их PC AT «московский стандарт» — столь же достойный повод для рассказов о трудном отрочестве, как и ЕС-1022 нашей молодости, пришедшейся на конец семидесятых. Бог с ними. В конце концов, слабая техника — не более чем стимул к самосовершенствованию. Следует помнить, что ОП — это прежде всего люди.

Об этих людях хочется петь. Редко кого из них учили программировать. Так, наш доцент утверждал, что OPEN FILE в цикле ведет к рывкам диска то по часовой стрелке, то против нее.

Таким образом, мы учились сами, и это позволяет мнить себя архангельскими мужиками. Все, что знают и умеют представители ОП, почерпнуто главным образом из поучительной истории собственных заблуждений (первым было то, что компилятор является устройством).

Отсутствие кузниц кадров привело к пестроте наших рядов. Автору известны программисты, в предыдущем существовании бывшие строителями, врачами, юристами, штурманами дальнего плавания. Есть даже один выпускник МИУ им. С.Орджоникидзе.

Поэтому, если с техникой и примкнувшим к ней системным программированием наше дело обстояло и обстоит не блестяще, то отечественной научной мысли, когда она набредает на программирование, есть что вспомнить. Об академике Глушкове спустя век будут говорить не меньше, чем о Мальтусе. Действительно, десять в четырнадцатой степени расчетов год — это вам не фунт изюма. Это нечто сродни российским просторам. Это верно и потому всесильно, это в Беловежской пуще не отменишь. В силу этого ОП необходимо и будет жить.

Оно будет жить еще и потому, что условия для такого рода деятельности у нас всегда были самыми благоприятными, то есть, по терминологии Ф.Кана, «варварскими» (под этим принято понимать старый гараж и холодный кофе с черствыми пирожками; понятно, что отечественные реалии превосходят подобное представление). А кроме того, где еще можно продать дешевую поделку на Clipper или, того лучше, просто PKZIP с «Лексиконом» за те же деньги, что разработчик какой-нибудь экспертной системы зарабатывает за год? ОП, таким образом, есть самоценный процесс, лишенный и тени меркантильности.

И после этого респектабельные журналы попрекают нас устами своих напыщенных авторов: мы, дескать, не платим за компиляторы! Почему бы этим моралистам не прокомментировать стодолларовые оклады тех из нас, кто работает на иностранных работодателей? Кто поумнее, тот понимает, что происходит на заснеженных просторах России, и им воздается. Недаром у нас Borland любят почти так же, как «Спартак».

Что день грядущий? Несмотря на интегрированные среды и турбореактивные компиляторы, жизнь людей, посвятивших себя ОП, не становится легче. Отечественный пользователь уверовал в компьютеризацию и оседлал отечественного программиста. Последнему не позавидуешь, поскольку у нас нет и двух контор, в которых при ведении дел соблюдалось бы похвальное единообразие. Не дает скучать и государство — правила налогообложения в РФ алгоритмизации не поддаются. Из-за инфляции приходится еженедельно увеличивать разрядность переменных и ширину граф в отчетах. Поэтому написание хорошей программы бухучета в условиях ОП становится проблемой искусственного (противоестественного) интеллекта. В этих условиях никто, кроме нас, нам не поможет.

Итак, да здравствует ОП, и сохраним в первозданной чистоте его сакральные традиции! Любой ценой — продолжением выпуска ЕС-1840, отменой пенсий для программистов, президентским указом о применении исключительно основной кодировки ASCII. Иной путь, как то: выдача IBM PC в обмен на ваучеры и принятие государственного плана компьютеризации России — это путь к угасанию ОП, постепенному превращению нас в племя безликих клерков.

Жаль, что в россыпи отечественных профессиональных праздников не нашлось место для дня программиста. Ну да ладно. Были бы мы, а день будет. IF будет день THEN будет пища. ■

ARS MOSCOW

HEWLETT PACKARD
Authorized Wholesaler

НАДЕЖНОСТЬ, ПРОВЕРЕННАЯ ВРЕМЕНЕМ

ПРИНТЕРЫ		КОМПЬЮТЕРЫ	
HP LaserJet 4L	–$ 790	HP Vectra 386/33N	–$ 1.380
HP LaserJet 4	–$ 1.690	HP Vectra 486s/33M	–$ 1.790
HP DeskJet 510	–$ 360		
HP DeskJet 550C	–$ 750	СКАНЕРЫ	
HP DeskJet Portable	–$ 360	HP SkanJet IIp	–$ 910
HP PaintJet XL 300	–$ 2.950	HP SkanJet IIc	–$ 1.660

НЕМЕДЛЕННАЯ ПОСТАВКА СО СКЛАДА
ГАРАНТИЯ HEWLETT-PACKARD

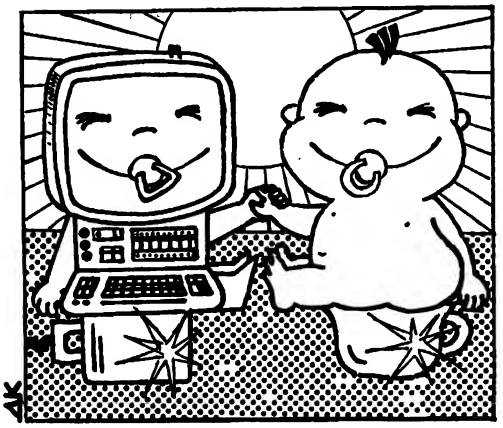
113035 Москва, ул. Осипенко, д. 15, корп. 2 офф. 207
Тел.: 237-66-81; 230-56-12; 220-27-59; Факс: 230-21-82; Телекс: 412417 SVET SU

ВЕСЬ СПЕКТР ОБОРУДОВАНИЯ HEWLETT-PACKARD

Журнал «Монитор» можно приобрести в городах России, Украины, Беларуси, Узбекистана, обратившись к нашим официальным дилерам.

1. Самара, МП «Сети», директор Дмитрий Грецов
Тел. (8462) 354-932
2. Волгоград, арендное предприятие «Техническая книга»
Тел. (8442) 363-597
3. Чебоксары, фирма «Демос»
Тел. (8352) 208-395, 426-492, факс 209-437
4. Киев, МП «Диас»
Тел./факс (044) 220-77-15, тел. 518-80-24
5. Днепропетровск, Попов А.Е.
Тел. (0562) 428-593
6. Минск, НПП.«Триумф»
Тел. (0172) 666-335, факс 666-606
7. Ташкент, Центр молодежных исследований
Тел. (3712) 688-847
8. Тамбов, АО «От Елены» ЛПС «Shift»
Тел. (0752) 22-44-63
9. Пермь, Попов А. Л.
Тел. (3422) 33-30-91
10. Иркутск, Мадьяров И. И.
Тел. (3952) 46-19-68

Языки программирования четвертого поколения



А. Штрик

Часть 1. Общая характеристика

Предпосылки появления языков четвертого поколения

Долгое время считалось, что для выполнения плановых сроков разработки сложных программных проектов достаточно увеличить количество программистов. Однако практика показала, что лишь в очень немногих случаях подобный подход позволял решить эту проблему. Поэтому возникла необходимость поиска новых методов и автоматизированных средств создания программ, благодаря которым удастся повысить производительность имеющегося штата программистов, не увеличивая их числа. Перечень некоторых таких методов и их распространенность (в процентах к числу респондентов) приведены на рис. 1; сплошные линии соответствуют распространенности на начало 90-х годов, пунктирные — прогноз на 1995-96 годы по материалам [1]. Из этих данных видно, что языки четвертого поколения занимают одно из важных мест в арсенале методов повышения производительности.

Языки четвертого поколения возникли на базе инструментальных систем программирования для больших компьютеров и мини-ЭВМ и первоначально предназначались для профессиональных программистов — разработчиков прикладных систем. Основными особенностями этих средств были: наличие непроцедурного языка очень высокого уровня, реализующего замкнутый набор функций в данной прикладной области; поддержка достаточно широкого набора системных функций (утилит); возможность использования языков на нескольких аппаратно-системных платформах; поддержка стандартных форматов хранения данных (интерфейс с БД); удобный интерфейс с пользователем. Эти языки давали многократный рост производительности по сравнению с языками третьего поколения (такими как COBOL, Algol), однако область их прикладного применения в первое время была сильно ограничена. Появление персональных ЭВМ, высокопроизводительных программных и системных средств для персональных ЭВМ и сетевых средств, автоматизированных систем поддержки программной инженерии (CASE технологий) и искусственного интеллекта позволили резко расширить область применения языков четвертого поколения, круг решаемых с их помощью прикладных задач и превратить эти языки в эффективный метод повышения производительности программистов.

Эволюция развития языков программирования и их особенности

В процессе развития в языках программирования сменилось несколько поколений, каждое из которых характеризовалось качественными

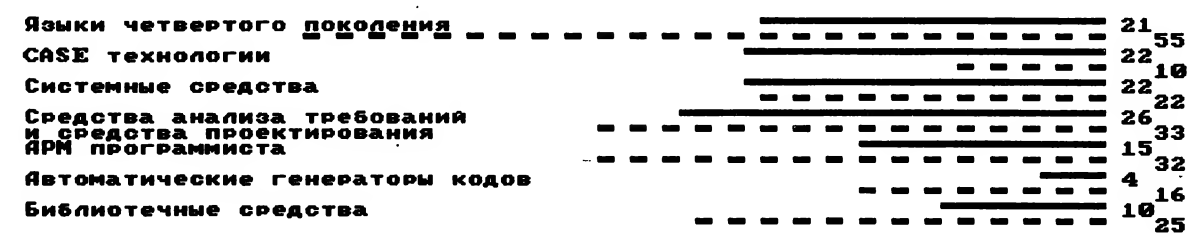


Рис. 1

изменениями и сопровождалось увеличением на порядок производительности создания программных систем.

В языках первого поколения программирование велось в машинных кодах и имело характер сложной комбинаторной задачи, так как одновременно с составлением команд программисту нужно было также распределить память.

Принципиально важным усовершенствованием процесса программирования явилось введение символических адресов, позволивших разделить процессы составления команд и распределения памяти. Память стала распределяться после составления программы в символических адресах путем автоматической замены буквенных символов числовыми значениями. Это привело к возникновению так называемых автокодов, или языков символического кодирования, которые и стали называться языками второго поколения.

С дальнейшим развитием вычислительной техники возникли проблемно-ориентированные языки, целиком ориентированные на особенности решаемых задач и не зависящие от конкретной машины (например, Fortran, COBOL, PL/1, C, Pascal). Эти языки, относимые уже к третьему поколению, сделали значительный шаг навстречу пользователям. В них используются английская символика и математические выражения в форме, близкой к обычной. Тем не менее программирование на языках третьего поколения осталось сложной, требующей профессиональной подготовки работой, сопряженной с большими временными затратами из-за сложности описания всех деталей процесса решения поставленной задачи. Особенно много усилий тратится на отладку программ и устранение ошибок, неизбежных при написании программ.

Последние годы характеризуются появлением новых средств общения человека с ЭВМ. Вместо традиционного составления программ стали использоваться непроцедурные методы, при которых определяются виды данных, извлекаемых или вводимых в базу данных, и необходимые преобразования над этими данными, а также графические средства разработки программ. Подобные методы предполагают, что пользователям предоставляется иерархический ряд меню, а программисту — высокоуровневые командные языки, которые назывались по-разному: декларативными языками, языками ограничений, высокопродуктивными языками, языками сверхвысокого уровня. В 1985 году Дж. Мартин использовал для их обозначения термин 4GL — Fourth Generation Languages (языки четвертого поколения), который отразил фактическую смену поколений компьютерных языков и стал общепринятым.

В настоящее время под языками четвертого поколения подразумеваются непроцедурные языки с нотацией, близкой к естественному языку, которые позволяют пользователю быстро и наглядно формировать алгоритм работы и специфицировать желаемые результаты работы проектируемой программной системы. Как и языки предыдущих поколений, языки четвертого поколения предназначены для разработки прикладных программ, и их главной целью является стремление максимально повысить производительность разработки программ и снизить количество ошибок в той прикладной области, для которой создается язык. В отличие от традиционных языков, языки четвертого поколения имеют более высокий уровень, поскольку включают в себя средства проектирования и решают эту задачу на основе новых методологий и средств, обеспечивающих дружественность компьютерной системы по отношению к пользователю. Второй отличительной чертой 4GL является то, что они ориентированы не только на профессиональных программистов, но и на конечных пользователей из различных прикладных областей, поскольку предоставляют им мощные и удобные средства для работы с прикладными системами и даже позволяют привлечь конечных пользователей к разработке своих систем.

На рис. 2 (в процентах к числу респондентов) показано, что в США в настоящее время языки 4GL можно считать вторым по популярности классом языков программирования; эксперты даже ожидают, что языки 4GL могут стать в некоторых приложениях более распространенными, чем традиционные системы программирования.

Все языки четвертого поколения различаются прикладными областями и функциональными возможностями. Большинство промышленно используемых языков четвертого поколения в зарубежной практике ориентированы на создание информационных систем для приложений в области бизнеса и используются совместно с различными СУБД. Имеются, хотя и в значительно меньшем количестве, языки 4GL для разработки управляющих систем, работающих в реальном масштабе времени.

Помимо функциональных различий, эти языки отличаются аппаратной платформой и операционной средой, в которой они работают, а также наличием интерфейсов с различными СУБД, причем многие из языков 4GL поддерживают работу на нескольких платформах, с разными операционными системами и базами данных.

В итоге в индустрии программирования сейчас имеется множество языков и систем программирования четвертого поколения: от языка запросов SQL до систем серии dBase для ПЭВМ; только в США по данным на 1990 год их насчитывалось более 70; при этом упомянутая совокупность языков работает более чем с 25 различными операционными системами и имеет интерфейсы более чем с 30 СУБД. Фактически каждая крупная фирма, занимающаяся разработкой прикладного ПО, имеет свой язык. Правда, большинство из них представляет собой комплексы, работающие со строго определенной базой данных, а не традиционные инструментальные системы программирования.

Основные характеристики языков 4GL

Базовые принципы 4GL

Среди множества свойств, характеризующих 4GL, основными являются два, отражающие главные цели

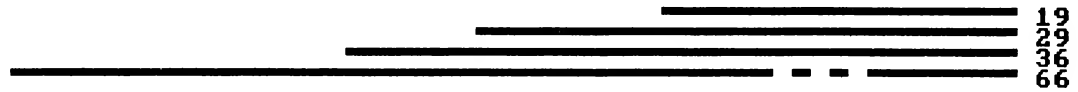


Рис. 2

языков 4GL и отличающие их от языков предыдущих поколений. Первый состоит в том, что языки четвертого поколения предоставляют пользователю возможность строить свою прикладную систему из крупных готовых блоков, реализующих функции, характерные для данной предметной области, тем самым значительно увеличивая производительность разработки и снижая количество ошибок. Примерами таких «строительных блоков» для создания информационных систем являются следующие функции:

определить структуры данных;
определить меню;
определить экраны/окна;
определить отчеты/документы;
определить логику работы программы.

Второй базовый принцип языков 4GL — стремление сделать человеко-машинное взаимодействие максимально дружественным человеку, эффективным и гибким. При этом существенно то, что это стремление продиктовано не просто соображениями удобства, а отражает новый этап развития информационной техники и технологии.

Функциональные требования

Исходя из задач эффективного выполнения целевого назначения, экспертами сформулирована базовая совокупность функциональных требований, которым должны отвечать языки 4GL; эта совокупность содержит, в частности, следующие требования к языкам:

- включать интегрированное множество функций с общим командным синтаксисом, в том числе основные функции создания прикладных программ, а также такие средства, как язык запросов, средства поддержки принятия решений, средства моделирования и прототипирования, средства статистического анализа, средства управления проектом и так далее;
- содержать хорошо структурированный язык очень высокого уровня;
- обеспечивать удобный и дружественный пользователю интерфейс;
- обеспечивать автоматическое документирование функциональных спецификаций;
- включать средства анализа и моделирования данных;
- обеспечивать высокую эффективность при сильной загрузке и поддержку обработки транзакций;
- обеспечивать совместимость с каким-либо языком третьего поколения в выбранной прикладной области и возможность включения программ, написанных на этом языке;
- реализовывать поддержку стандартных СУБД.

Дружественный интерфейс

Эффективность языков четвертого поколения и рост на порядок производительности разработки программ в большой мере определяется тем, насколько полно в них реализовано требование дружественного интерфейса, то есть теми свойствами языка, которые создают пользователю удобное окружение. Далее перечислены свойства языков 4GL, практически реализующие понятие «дружественный интерфейс»:

- использование естественных языковых конструкций и структур меню, понятных и удобных для пользователя, причем не требуется запоминать «компьютеризованный» синтаксис, последовательность команд или программ;
- обеспечение «интуитивного» уровня взаимодействия с компьютером, не требующего длительного обучения, при этом набор операций для профессионального специалиста в прикладной области должен быть столь простым, чтобы его можно было освоить за несколько часов;
- предоставление возможности выбора для работы набора команд, ориентированного на различный уровень подготовки пользователя (то есть возможности выбора естественного языка, двумерных структур меню, командного языка, близкого к естественному, пиктограмм, графических диаграмм, отчетов по образцам и так далее);
- предоставление для каждого уровня пользователя адекватных возможностей в части меню, запросов, помощи и самообучения;
- освобождение пользователя от необходимости решать проблемы низкоуровневого синтаксиса при ориентации на высокий уровень для разработки своего приложения;
- обеспечение необходимой скорости диалогового взаимодействия;
- организация диалога с использованием всей двумерной поверхности дисплея, использование современных средств организации интерфейса (пиктограммы, «мышь», раскрывающиеся меню, всплывающие окна и так далее);
- минимизация использования клавиатуры;
- применение «интеллектуальных» средств (редактор, обладающий интеллектуальными возможностями, подбор значений по умолчанию, поддержка пользователя в применении языка).

Взаимосвязь языков четвертого поколения и CASE

По существу языки 4GL должны рассматриваться как системы, представляющие собой ограниченные по функциональному назначению формы интегрированного окружения, применяемого при создании прикладных программ. Такое окружение включает значительно более широкий набор средств разработки и отладки программ по сравнению с обычными языками третьего поколения (например, диалоговые, графические, для работы с БД), а также средства поддержки других этапов жизненного цикла программных систем (например, проектирования). Анализируя эволюцию языков программирования с позиции обработки данных, легко видеть, что первое поколение языков манипулировало с битами (машинные языки); второе поколение работало с байтами и словами (ассемблеры); третье поколение оперировало с записями в массивах и файлах данных. Четвертое поколение обязано своему появлению базам данных: необходимость создания и развития мощных прикладных языков для определения данных и манипуляций с ними привели к тому, что к языкам систем управления реляционными базами данных добавились инструментальные средства облегчения прикладной разработки.

По существу языки 4GL могут рассматриваться как ограниченная форма интегрированного инструментального окружения разработки прикладных программ, включающее СУБД (программные средства хранения и сопровождения словаря данных), текстовый и графический редакторы, средства управления разработкой, а также другие базовые средства. Главными компонентами таких систем для современных языков четвертого поколения являются:
- язык очереди базы данных, обычно это язык SQL для запросов в базе данных и манипуляции с данными;
- процедурный язык третьего поколения (чаще всего язык С) для.более сложного манипулирования данными;
- генератор отчетов — простой язык для поиска сложных данных и формирования изображения результата вычислений на экране;
- экранный генератор для обеспечения экранного интерфейса с базой данных.

Аппаратно-системная реализация современных языков (а фактически инструментальных окружений) четвертого поколения должна удовлетворять следующим требованиям:
- обеспечивать функционирование на основных типах ЭВМ и в различных сетевых конфигурациях в зависимости от потребностей пользователей;
- предоставлять возможность распределенного взаимодействия множества конечных пользователей с поддержкой доступа к центральной базе данных;
- включать интерфейс с другими инструментальными средствами создания и сопровождения программных систем, чтобы обеспечить поддержку полного жизненного цикла разработки программной системы, включая начальные стадии.

На практике, однако, не все из перечисленных требований реализованы в каждом из языков класса 4GL. Кроме того, помимо перечисленных требований, конкретные языки могут соответствовать и другим требованиям, связанным с определенной областью приложения языка.

Эффективность языков четвертого поколения

По некоторым оценкам, при использовании языков 4GL производительность программирования у специалистов высокой квалификации может вырасти до десяти раз по сравнению с традиционными методами. Исходя из того, что средний программист, работающий на COBOL, согласно имеющимся данным, пишет (с учетом правки и редактирования) около двадцати кодовых строк в день, его производительность при работе с языками четвертого поколения может в эквивалентном исчислении составить до 100-200 кодовых строк в день.

Высокая эффективность является главным стимулом развития и распространения языков четвертого поколения. Однако эффективность их использования существенно зависит от характера приложений. Она возрастает в тех приложениях, где содержится много текстов, необходима сложная обработка больших массивов данных, возможно использование графических средств. Специалисты на основе практического опыта работы выделяют ряд факторов, важных для достижения высокой продуктивности применения языков четвертого поколения:
- максимальное использование непроцедурных функций;
- использование графических средств для автоматического проектирования;
- использование диаграммной техники для создания спецификаций и программ;
- доступ в базу данных в режиме реального времени;
- быстрое моделирование;
- интерактивный режим проектирования;
- автоматическое документирование;
- модульное программирование.

В числе прочих факторов, влияющих на производительность труда программиста, отмечается необходимость высокого уровня организации работы пользователей. В частности, при правильном выборе системных средств применительно к конкретной области приложений и должной организации использования этих средств пользователю не нужно быть программистом высокой квалификации для достижения хороших результатов. В то же время пользователю необходимы эффективные средства обучения и адаптации к новым системам.

Языки 4GL в жизненном цикле разработки программ

Так же как и CASE средства проектирования программ, которые направлены на интегрированную автоматизацию полного жизненного цикла, языки 4GL поддерживают основные этапы проектирования, кодирования и тестирования. Основное различие между языками 4GL и CASE средствами заключено в числе реализуемых на каждом этапе функций и уровне их автоматизации. При этом языки 4GL, начав свое развитие с конца 70-х годов, в настоящее время достигли пика своего расцвета, в то время как CASE средства, появившись на десять лет позже, находятся на пути интенсивного подъема. В связи с этим специалисты считают, что основная тенденция дальнейшего развития обоих направлений состоит в их интеграции: для языков 4GL это является продолжением эволюции, а для CASE средств это расширение сферы влияния.

В зависимости от функционального назначения и области применения существующие языки 4GL делятся на несколько функциональных категорий.

Простые запросы: с этой категорией большинство программистов знакомо, поскольку фактически этот тип языка существовал до появления термина 4GL.

Сложные запросы и корректировка: речь идет об обслуживании совокупности одновременно поступающих запросов, причем на несколько разных типов объектов, кроме того, здесь же предусматриваются возможности корректировки, которые, однако, в большинстве случаев являются внутренней составляющей СУБД и включаются как составная часть продукта 4GL.

Генераторы отчетов — это одна из наиболее важных сфер применения языков 4GL, хорошие генераторы должны поддерживать процесс навигации по базе данных, когда требуемые данные находятся в одной или нескольких записях; важной стороной является

возможность осуществления форматирования документов и других видов работы с ними.

Языки поддержки решений: эти средства должны обладать рядом отличий от традиционных языков, предусматривая реализацию таких функций, как, например, анализ «что, если?» или статистический, либо прогнозный анализы.

Генераторы прикладных приложений — эта категория языков особенно интересна разработчикам прикладных программ. Она содержит наиболее мощные языки, которые объединяются с серией средств, обеспечивающих полную разработку сложных программных систем. Наряду с этим они включают множество дополнительных средств, в частности, средства проверки достоверности данных, корректировки записей, возможность обработки данных программами, написанными на языках третьего и четвертого поколений и так далее.

Все перечисленные категории языков четвертого поколения специалисты разделяют на три класса в зависимости от наиболее вероятных классов пользователей:

- языки для манипулирования и управления данными в реляционных СУБД;
- языки 4GL для разработчиков прикладных приложений;
- языки 4GL для применения конечными пользователями в информационных системах и центрах.

На рис. 3 для каждого из классов приведен ряд известных представителей языков; их основные характеристики и возможности будут рассмотрены в следующем разделе.

На рис. 4 показано, как меняется структура жизненного цикла разработки программ при использовании традиционных языков и языков четвертого поколения. Качественное изменение характерно для начальных этапов создания прикладной системы; следует также иметь в виду и количественные изменения технико-экономических показателей разработки программных систем, о чем шла речь ранее. ■

Литература
1. How Europe is using CASE// Datamation.– August 1, 1990.

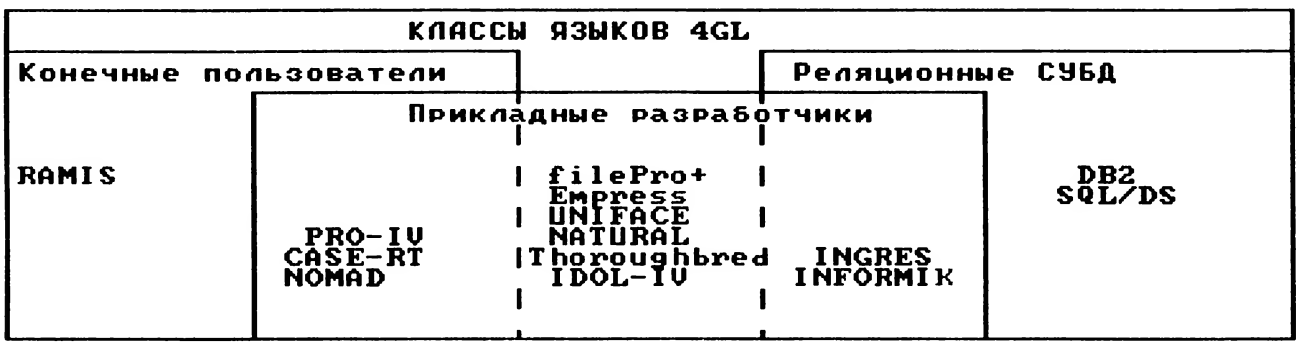


Рис. 3

Рис. 4

фирма ЭКСИМЕР

Мы предложим Вам современные компьютеры
на основе микропроцессоров
фирмы intel.

excimer personal computer

Винчестеры различной емкости

Caviar

WESTERN DIGITAL

Видеоакселераторы

PARADISE FOUND HERE
PARADISE.

COMPEX®

Сетевое оборудование фирмы
Сетевое программное обеспечение

NOVELL®
Past, Present and Future of Network Computing

Факсимильные аппараты различной сложности
фирмы DAEWOO
DAEWOO CORPORATION

Всемирно известные принтеры

EPSON®
SEIKO EPSON CORPORATION

star

HEWLETT PACKARD

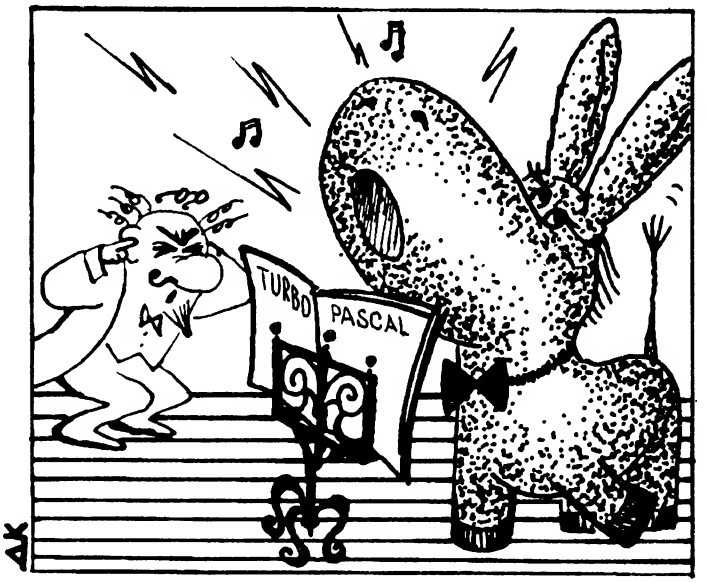
Копировальные аппараты

Canon

А так же многое другое
оборудование.
Мы предлагаем гарантийное и послегарантийное
обслуживание техники. Установка и обслуживание
локальных и глобальных сетей
Наш адрес: г. Москва, ул. И. Бабушкина, д.23, к.1
Тел: (095) 125-7001 Факс: (095) 125-7868
Сетевой отдел тел/факс: (095) 319-7418
Факс дилерского отдела: (095) 319-7490

Специфические ошибки Turbo Pascal

М. Трофимов



В этом нет ничего такого, чего каждый опытный программист не знал бы в общем, с точки зрения здравого смысла.

Из дискуссии

Разговор о специфических ошибках Turbo Pascal фирмы Borland Int. полезно начать с двух ссылок на его документацию: инструкцию к Turbo Debugger [1], [2] и руководство программиста [3]. Последнее содержит полный список отклонений языка Turbo Pascal от стандартного Pascal; как будет видно из дальнейшего, эти отклонения во многих случаях и являются источником наиболее неприятных специфических ошибок. Однако «фирменный список» таких ошибок далеко не полон (да и возможно ли сделать исчерпывающий список подобного рода?). Здесь мы не будем говорить об объектно-ориентированных расширениях версий 5.5 и 6.0 Turbo Pascal. Не обсуждается и новая версия Turbo Pascal — версия 7.0, появившаяся уже после написания этой статьи. Судя по предварительным отзывам, эта версия совершеннее предыдущих. В таком случае, не теряет ли смысла предлагаемое изучение специфических ошибок? — Очевидно, нет, поскольку наша цель не в наклеивании ярлыков на тот или иной инструмент («хороший» – «плохой»), а в попытке выявления специфики одного из наиболее распространенных диалектов языка Pascal с точки зрения подверженности ошибкам. Усовершенствование инструментальной оболочки при сохранении принципов самого языка, по-видимому, не может полностью устранить возможности появления какого-либо типа специфических ошибок, в лучшем случае эта ошибка перейдет из разряда «труднообнаруживаемых» в разряд «легкопреодолимых».

За основу можно взять следующую простую классификацию, схематически представленную на рис. 1. Элементы схемы пронумерованы: (1)...(8), и по ходу изложения мы будем ссылаться на них. Например, рис. 1(2) — ошибки, специфические для языка Pascal, о которых следует упомянуть прежде всего. Для этого рассмотрим следующий фрагмент [2]:

```
for i := 1 to 10 do;
   writeln (i);
```

Допущенная ошибка вызвана тем, что при наборе текста программы за ключевым словом do ошибочно была поставлена точка с запятой. Неприятности в результате такой небрежности ожидают программиста вне зависимости от того, работает ли он на Turbo Pascal или пользуется другим компилятором языка Pascal. Такая ошибка не будет специфичной для ряда языков программирования с алголоподобным синтаксисом, где каждое «do» требует симметричного закрывающего «od». А вот ошибки, вызванные тем, что цифра «1» и буква «l» имеют похожие очертания, не обусловлены исключительно спецификой конкретного языка программирования (рис. 1(1)), или, если хотите, специфичны для всех языков, в том числе и естественных, где применяется латинский алфавит, и не специфичны, например, для русского.

Особый интерес представляют ошибки, специфические для Turbo Pascal, но не характерные для классического Pascal (рис. 1(7)). Многие из них стали возможны из-за нарушений принципов защитного программирования в концепции Turbo Pascal. Например:

```
var
   i,j : integer;
begin                {тело программы}
   i := j+1;
```

В единственном операторе этого фрагмента происходит увеличение на единицу значения переменной j и присваивание его переменной i, а поскольку данный оператор будет выполняться первым, значение j не определено.

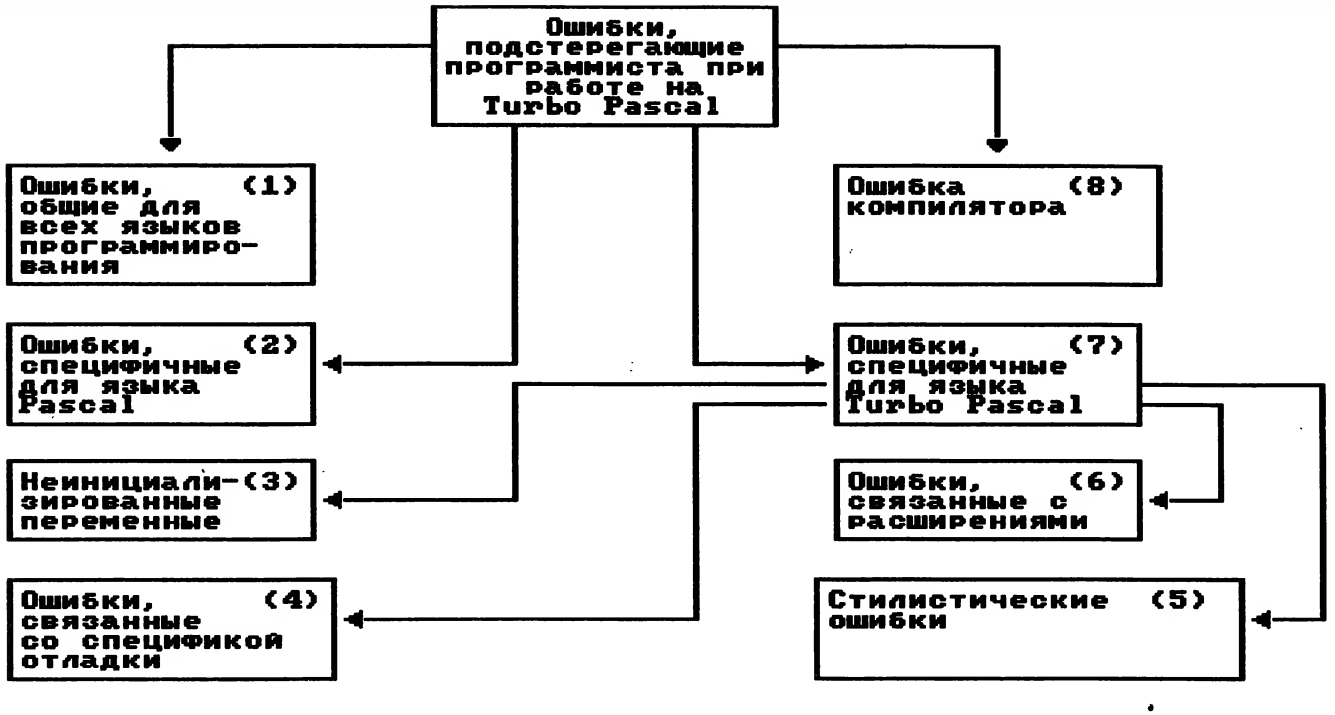


Рис. 1. Классификация ошибок

В соответствии с концепцией защитного программирования попытка использования неопределенного значения переменной должна детектироваться системой исполнения (run-time system) как ошибка исполнения (run-time error). К сожалению, система исполнения Turbo Pascal не отслеживает такие ошибки (рис. 1(3)). Согласно распределению памяти, из области, соответствующей переменной j, берется значение, которое к этому моменту там находилось, и интерпретируется как целое число типа «integer», затем к нему прибавляется единица, и результат помещается в область, соответствующую переменной i.

Особенно много неприятностей подобные ошибки могут причинить программистам при использовании переменных ссылочного типа. Я сам как-то наблюдал ситуацию, когда в некоторой программе при наборе текста была пропущена строка

```
list^.next := nil;
```

Построив список list, программа перешла к выполнению фрагмента

```
while (list^.next <> nil) and
      (list^.date <> 0) do
         list := list^.next;
list^.date := 1;
```

И, всласть напутешествовавшись по памяти (первый оператор этого фрагмента), программа наконец наткнулась на область, воспринятую ею как запись с полем date, равным 0. В полном соответствии с условием цикл while был закончен, и значение, прочитанное как ноль, было изменено на 1.

После возврата в режим редактирования интегрированной оболочки Turbo Pascal программист подал команду «сохранить файл», нажав F2. Однако ему не повезло, потому что при обращении к винчестеру оказалась задействованной та самая область памяти, которая как раз и подверглась некорректному изменению, описанному выше. В результате таблица размещения файлов (FAT) винчестера оказалась разрушенной!

Очень неприятная ошибка, также (в отличие от стандарта) недетектируемая в Turbo Pascal, — это изменение счетчика цикла for в теле цикла. Например, следующая программа зацикливается:

```
program WrongLoop;
var
   i,j : integer;
begin
   j := 1;
   for i := 10 downto 1 do
   begin
      writeln (i:6, j*i:6);
      inc (i);                    {Должно быть inc (j)}
   end;
end.
```

Несмотря на то, что концепция защитного программирования органично вошла в Pascal при его создании, а впоследствии была закреплена стандартами, она проработана далеко не во всех деталях. Подробному обсуждению практических и теоретических проблем обнаружения неинициализированных переменных была посвящена, в частности, работа [5]. Вот всего лишь один вопрос: нужно ли считать ошибкой

присвоение переменной-записи (record), если одно из полей этой записи не инициализировано:

```
var
   coord1, coord2 :   record
                        x,y : real
                      end;
begin
   coord1.x := 3.14;
   coord2 := coord1;
```

Неудивительно, что сложность практического воплощения принципов защитного программирования с учетом конкурирующих требований (прежде всего, оптимизации кода как самого компилятора, так и генерируемых им программ) заставляет многих изготовителей программного обеспечения, а не только фирму Borland, подвергать ревизии принципы стандарта.

Несколько слов нужно сказать и о специфике отладки; кстати, именно с этим связана трудность в определении местоположения некоторых типичных ошибок (рис. 1(4)).

Отладка программы в интегрированной среде (ИС) Turbo Pascal представляет собой работу непосредственно с исходным текстом, тогда как классическая схема представляет собой серию прогонов отлаживаемой программы с последующей выдачей листинга, по которому программист и исправляет исходный текст. Исключение листинга в Turbo Pascal выглядит как значительное достижение. К сожалению, вместе с листингом ушла и очень полезная для программиста служебная информация, облегчающая чтение и анализ программы. Так, в типичном для текстов на Pascal фрагменте

```
            end
         end
      end
   end
end
```

очень легко ошибиться и пропустить одну из скобок «end». Традиционная форма листинга включала колонку с уровнем вложенности скобки, формируемую транслятором, которая помогала быстро найти ошибочное место в программе. В Turbo Pascal локализовать подобную ошибку гораздо труднее. Конечно, классический «бумажный» листинг — не единственная форма представления служебной информации; например, иной подход (гиперлистинг) был использован в анализаторе текстов на языке Pascal hyper-Rainbow [4].

Цикл поиска синтаксических ошибок для ИС Turbo Pascal состоит из:

1) компиляции от начала (первая строка исходного текста) до первой синтаксической ошибки;
2) прерывания компиляции и сообщения об ошибке;
3) исправления этой ошибки пользователем;
4) возврата к шагу 1 до тех пор, пока синтаксические ошибки встречаются в исходном тексте.

Компиляторы, работающие в пакетном режиме, действуют в соответствии с принципами восстановления процесса компиляции при синтаксических ошибках: в результате первого прогона программист получает листинг с полным списком ошибок (если, конечно, компилятор не натолкнулся на слишком грубую «фатальную» ошибку, прервавшую процесс компиляции). Для исправления разнородных ошибок, обнаруженных в пакетном режиме, программисту придется совершить много лишних движений, корректируя каждую из них, да еще затратить дополнительные усилия на поиск следующей по листингу. В диалоговом же режиме курсор автоматически устанавливается на место ошибки в исходном тексте. Правда, когда однородных ошибок много, для программиста будет полезнее полный список их, чем сообщение только о первой встреченной ошибке. Например, программист определил переменную «next_i», и первая часть исходного текста содержит правильное имя этой переменной. Вторая часть исходного текста содержит неправильное имя — «i_next» — для той же самой переменной. В этом случае все неправильные имена могут быть скорректированы с помощью одной-единственной команды контекстной замены «i_next» на «next_i», нужно только знать, что ошибка распространена, то есть иметь перед глазами полный список ошибок в тексте программы. Особенно часто многократный повтор одной и той же ошибки получается при слиянии двух программных текстов в один.

Похожий на предыдущий и не менее распространенный случай — это путаница параметров в вызовах процедур. Предположим, Вы написали собственную процедуру вычерчивания окружности:

```
procedure circle   (Radius {радиус},
                    x0, y0   {координаты
                              центра} : real)
```

и поместили ее в модуль MyGraph. В программе Вы перепутали обращение и написали

```
circle (x,y {координаты центра}, r {радиус});
```

Поиск этой ошибки может стоить Вам немалых усилий, особенно если Вы работаете с уже отлаженной библиотекой MyGraph.TPU. К сожалению, склонность к подобной путанице имеют и многие другие современные языки программирования. Как тут не вспомнить о старом ассемблере для IBM/360 с его ключевыми макропараметрами:

```
circle x0 = x, y0 = y, radius = r
```

(Громоздко, но никакой путаницы!)

Ряд ошибок может быть спровоцирован размером фрагмента, слишком большого для того, чтобы целиком уместиться на экране. Вот один из возможных случаев (строки пронумерованы, и часть их опущена как несущественная):

```
{200}   (* for i:=1 to 5 do
   ...
{300}   *)
```

Иными словами, сто строк исходного текста забито комментариями — иногда это вполне оправданно. Вся беда в том, что, переместив экранную область редактора куда-нибудь в район 250-й строки, мы имеем

полную возможность воспринимать данный текст как работающий, то есть порождающий код, начисто забыв о его «закомментированности». Похожие проблемы возникают и в случае слишком объемных составных операторов. Так, имея перед глазами строку

```
x := 5;
```

можно подозревать, что несколькими десятками строк ранее стоит

```
with XYZ do
  begin
```

Можно также предположить,что речь идет о присвоении значения 5 полю x:

```
XYZ.x := 5;
```

Возможно, перед оператором «with» стоит

```
if XYZ.x = 0 then
```

Тогда данный оператор будет выполняться только по достижении соответствующего условия. Особенно легко потерять ориентиры и «заблудиться» в исходном коде при отладке больших программ с помощью встроенного отладчика исходного текста.

Одной из разновидностей, разбираемых в [1], являются ошибки, возникающие при применении одинаковых имен для локальных и глобальных переменных. Этим ошибкам подвержен не только Turbo, но и стандартный Pascal, и их легко избежать, используя уникальные имена при написании программы. Имея в виду индивидуальный стиль программиста, такие ошибки можно назвать «стилистическими» (рис. 1(5)). Реализовать механизмы их обнаружения технически несложно. Детектор может быть как встроенным в компилятор, так и оформленным в отдельный функционально-независимый инструмент для анализа исходных текстов программ (такая идея и была реализована в уже упомянутом hyper-Rainbow).

Вообще следует отметить, что, как и в естественных языках, стилистические ошибки в языках программирования не всегда очевидны и часто спорны. К примеру, как уже говорилось, источником многих ошибок является использование глобальных переменных в процедурах и функциях, недаром ряд учебников призывает этого не делать. И тем не менее практически в любом сколь-нибудь объемном программном тексте на Pascal используются (пусть и ограниченно) глобальные переменные, в том числе и в программах очень известных авторов, например, изобретателя языка Pascal Никлауса Вирта.

Имеет смысл упомянуть также и о стилистических небрежностях, которые, с одной стороны, вроде бы и не являются ошибками, потому что не приводят к вредным и неверным действиям программы, однако с другой — усложняют и засоряют исходный текст, затрудняя его понимание, отладку и модификацию. Одной из наиболее часто встречающихся небрежностей являются так называемые «мертвые имена», то есть конструкции, присутствие которых в исходном тексте программы совершенно излишне. В простейшем случае это может быть переменная, описанная в разделе «var» и нигде больше не упомянутая. Избавиться от такого «мусора», с неизбежностью скапливающегося в процессе работы над программой, не представляет особого труда, достаточно воспользоваться простейшим генератором перекрестных ссылок: если в таблице перекрестных ссылок имя переменной упоминается всего лишь один раз, то, скорее всего, это «мертвая» переменная. Чаще, однако, задача поиска лишних имен и даже лишнего кода далеко не столь тривиальна. Так, вызов процедуры из библиотечного модуля может встретиться только однажды, а имена функций будут появляться по меньшей мере дважды: во-первых, в заголовке функции и, во-вторых, в ее теле — там, где функция возвращает значение:

```
program morten_names;
var
   i, j : integer;

function morten_func (i : integer) : integer;
begin
   morten_func := i * i
end;

begin
   writeln ('Квадраты чисел 1..10:');
   for j := 1 to 10 do
   begin
      i := morten_func (j);
      write (j, sqr (j))
   end
end.
```

При том, что проблема «мертвых имен» существует в большинстве языков программирования, по-видимому, в Pascal она может проявляться особенно остро, что объясняется рядом особенностей языка, например, требованием обязательного описания нестандартных имен до их употребления. В некоторых компиляторах с Pascal предусмотрена специальная диагностика «мертвых имен» (правда, как правило, довольно примитивная). В Turbo Pascal такой диагностики нет. Нет в Turbo Pascal и диагностики ошибок, связанных с проверками присваивания результата функции — речь идет о том, что если Вы пишете некоторую функцию, например

```
function A : integer;
   begin
```

то в конце тела функции обязательно должен стоять оператор присваивания, возвращающий результат функции:

```
      A := ...
end;
```

Стоит упомянуть еще о двух типах ошибок: обусловленных расширениями языка в системно- и аппаратурно-зависимом направлении (низкоуровневые возможности) и общих языково-независимых (алгоритмические ошибки). В отличие от ошибок, проис-

ходящих из-за отклонения Turbo Pascal от принципов стандартного Pascal, такие возможности, как низкоуровневые расширения, не предусматриваются стандартом. По сути дела, подобные ошибки специфичны скорее не для Pascal, а для программирования низкого уровня. В частности, в [2] приведен пример «зацикливания», вызванного тем, что переменная i типа word никогда не может иметь отрицательного значения и, значит, условие выхода из цикла i >= 0 всегда будет истинным.

Обижаться в таких случаях, как говорится, «не на что», поскольку известно, что низкоуровневое программирование особенно подвержено ошибкам, и, если уж программист отваживается на применение низкоуровневых средств, он должен быть готов ко всяческим «сюрпризам». Однако, к сожалению, Turbo Pascal может преподнести «низкоуровневый сюрприз» и в программах, не требующих ничего сверх обычных возможностей стандартного Pascal. Например, расширения Turbo Pascal допускают записи выражений, состоящих из логических функций (not, and, or) с целочисленными переменными. Ошибки в логических выражениях условных операторов (if, while, repeat) всегда довольно неприятны. Однако если при традиционном подходе выражение, начинающееся с

```
not i
(здесь i — переменная типа integer)
```

воспринимается компилятором как синтаксическая ошибка, то в Turbo Pascal выражение

```
not (i=j)        {выражение 1.}
```

и выражение

```
not i=j          {выражение 2.}
```

имеют смысл, что можно продемонстрировать следующей шуточной программой. (К сожалению, на практике такие «шутки» выглядят весьма печально.)

```
program strange;
var
   i, j : integer;
begin
   i := 1;
   j := 2;
   if not (i=j) {выражение 1}
      then writeln ('Ok: ', i, ' <> ', j)
      else writeln (i, ' = ', j, '?');
   if not i=j {выражение 2}
      then writeln ('Ok: ', i, ' <> ', j)
      else writeln (i, ' = ', j, '?');
end.
```

Помимо расширений, Turbo Pascal предоставляет в распоряжение программиста множество трюков. Некоторые из них даже приводятся в инструкции в качестве типовых приемов программирования. Так, для сравнения векторов инструкция [3] рекомендует воспользоваться следующей функцией с нетипизированными параметрами X и Y:

```
function eq (var X, Y; size : word) : Boolean;
type
   bytes = array [0..MaxInt] of byte;
var
   n : integer;
begin
   n := 0;
   while (n < size) and
         (bytes(X)[n] = bytes(Y)[n]) do
      inc (n);
   eq := n = size
end;
```

Пусть, к примеру, мы хотим сравнить два вектора, A и B, определенные как

```
var
   A  : TypeA;
   B  : TypeB;
```

Причем

```
type
   TypeA  = array [1..5] of integer;
   TypeB  = array [1..10] of integer;
```

В этом случае выражение

```
eq (A, B, SizeOf (TypeA));
```

обеспечит сравнение первых пяти элементов векторов A и B. А выражение

```
eq (A, B, SizeOf (TypeB));
```

очевидно, будет бессмысленным. Или, например, другой распространенный трюк. Пусть нам нужно читать строку

```
var
   line : string [80];
```

и укорачивать ее каждый раз после прочтения до восьми символов. Определим

```
var
   length : absolute line;
```

Теперь можно записать следующие операторы, исполняющие необходимые нам действия:

```
readln (line);
length := 8;
```

Все будет хорошо, пока нам не встретится строка короче восьми символов! Конечно, эти примеры даны здесь не для того, чтобы безоговорочно осудить трюки. Трюки при программировании для IBM PC бывают просто необходимы, но их реализация требует особого внимания. Кстати, еще одно замечание относительно последнего примера. Допустим, чтобы поправить ситуацию в случае слишком коротких строк, мы решили ввести проверку на длину строки, сказано — сделано:

```
if length (line) >= 8
   then
```

Но ведь мы уже использовали имя length в нашей программе, переопределив таким образом стандартную функцию length; кстати, line — тоже стандартное имя для графической процедуры. Turbo Pascal снабжен очень богатыми библиотеками, и случайно употребить какое-нибудь стандартное имя не по назначению не составляет труда. Впрочем, как и многие другие, ошибки этого типа легко устранимы, важно только выявить их.

В отношении алгоритмических ошибок кратко, по-видимому, можно сказать только то, что это большая разнородная группа, что они возникают еще на ранних стадиях разработки и детализации алгоритма и связаны с невниманием программиста к каким-либо факторам, например, к специальным случаям (исключительным ситуациям). Конечно, разные языки предлагают разные по выразительности способы описания исключительных ситуаций, однако, как правило, языковые средства имеют к таким ошибкам второстепенное отношение.

Кроме того, возможна ситуация, в которой Вы столкнетесь с действиями компилятора, по Вашему мнению, ошибочными (рис. 1(8)). Как быть в подобном случае? Здесь можно дать несколько общих рекомендаций, вне зависимости от того, имеете ли вы дело с Turbo Pascal или другим языком. Итак, прежде всего убедитесь в воспроизводимости обнаруженной вами «ошибки»: убедитесь, что и после «холодного рестарта» ситуация повторяется. Желательно также воспроизвести ошибку на другом компьютере. Затем минимизируйте код — сделайте минимальную по величине исходного текста программу, приводящую к той же ситуации. Помните, что вряд ли кто-нибудь сумеет Вам помочь, если Вы покажете ему программу в две тысячи строк, поэтому попытайтесь сделать что-нибудь попроще, скажем, в пару десятков строк, выкиньте все лишнее, не относящееся к Вашей ошибке.

Очень часто, проделав описанную работу, программист сам находит причину и устраняет ее. Если это Вам не удалось и ошибка все же возникает и в Вашей минимальной программе, задокументируйте ее, то есть сделайте полное и в то же время лаконичное описание Вашей проблемы. А далее возьмите еще раз инструкцию и листайте до тех пор, пока не убедитесь, что ничего подобного в инструкции не предусмотрено. Здесь надо особо подчеркнуть, что в данном случае следует работать именно с инструкцией, то есть официальным документом на продукт, а не с какой-нибудь другой книжкой об этом продукте, где интересующие вас подробности как раз и могут быть опущены по авторскому произволу. Только после этого обратитесь с Вашей проблемой к изготовителю или дистрибьютеру данного продукта. При этом лучше сначала сформулировать вопрос, не утверждая, что Вы нашли дефект и не высказывая претензий: это Вы всегда успеете сделать, если ответ не удовлетворит Вас или если Вы получите подтверждение, что найденная Вами «ошибка» — действительно ошибка продукта.

Приведенные рекомендации кажутся очевидными, однако почему-то многие программисты им не следуют, что отлично видно хотя бы по международным сетевым телеконференциям: очень часто задаются одни и те же вопросы, причем почти в тех же формулировках, что и разбираемые в инструкции примеры, как раз и отвечающие на эти вопросы.

Язык Pascal занимает особое место среди языков программирования. Он очень несложен, если судить по объему описания, составленного Н. Виртом. Это делает его легко стандартизируемым и очень полезным при обучении программированию. Общеизвестно, что сначала Pascal появился именно как «учебный язык», но вскоре, несколько неожиданно для автора, приобрел исключительную популярность у профессиональных программистов в качестве рабочего инструмента. Столь различные области применения языка неизбежно ведут к столкновению противоречивых интересов. То же самое переживает в подобной ситуации любой другой язык программирования, поэтому на примере развития Pascal и его диалектов можно выявить проблемы, общие для всех языков, как в настоящем, так, по-видимому, и в будущем. Одна из важнейших проблем — выявление специфических для данного языка ошибок и борьба с ними.

В заключение следует сказать о практических выводах, которые могут быть сделаны на основании анализа специфических ошибок. Было бы наивным отказываться от такого мощного и полезного инструмента, как Turbo Pascal. В то же время никто не запрещает использовать в своей работе наряду с Turbo Pascal и другие инструменты, и даже другие компиляторы. Тем более, что сейчас многие фирмы снабжают пакеты компиляторов специальными утилитами, обеспечивающими быстрый перевод исходных текстов на Turbo Pascal и обратно. Существует много инструментальных средств для борьбы со специфическими ошибками Pascal; одно из них — интерпретатор DrPascal американской фирмы Visible Software [6], [7]. ■

Литература

1. Turbo Debugger Users Guide, version 1.0.– Borland Intl., 1988.
2. Turbo Debugger Users Guide, version 2.5.– Borland Intl., 1991.
3. Turbo Pascal Reference Guide, version 5.0.– Borland Intl., 1988.
4. M. Trofimov. Simple Programming Tools May Be Useful Not Only for the Novices// Soft Market.– March 1992.– Special issue.
5. Willett Kempton & Brian A. Wichman. Run-Time Detection of Undefined Variables Considered Essential// Software — Practice and Experience.– April, 1990.– Vol. 20.
6. М. Трофимов. При отсутствии продукта. Демо-дизайн-II// Софт Маркет.– 1992.– № 16.
7. М. Трофимов. Как увидеть программу?// Мир ПК.– 1992.– № 2.

НТО

Софт·Москва

предлагает широкий спектр редакционно-издательских и полиграфических работ:

- литературное редактирование
- разработку рекламы для печатной продукции
- изготовление:

бланков, листовок, проспектов;
комплектов документации для программных продуктов;
сертификатов ценных бумаг;
визиток, наклеек, этикеток, приглашений;
слайдов;
изданий за счет авторов.

Наш адрес: 117419 Москва, а/я 765.
Тел.(095) 238-16-22, 943-25-53. Факс (095) 238-32-78

Профессиональная работа в среде Microsoft BASIC

Часть 3. Когда программы становятся большими

А. Колесов

Уважаемые читатели!

Ваши отклики на серию статей, посвященную программированию в среде Microsoft BASIC, показали, что у этого языка есть немало почитателей. Интерес к программированию в MS Basic привел к созданию в начале года информационно-консультационного центра «Ассоциация пользователей Microsoft BASIC» (ИКЦ АПМБ), с которым редакция «Монитора» тесно сотрудничает.

Результатом этого сотрудничества является, в частности, открытие в журнале, начиная с текущего номера, постоянной рубрики «MS BASIC».

Эта рубрика не будет последовательным курсом обучения программированию — для этого есть документация и методическая литература. Скорее она должна быть средством общения и обмена опытом между всеми, кто имеет отношение к языку BASIC: программистами-практиками, разработчиками языка, его продавцами и прочими. При этом мы не собираемся заниматься «ликбезом» — общая тематика будет подразумевать некоторую квалификацию программиста.

Кроме статей, важной частью рубрики будет анализ писем читателей и ответы специалистов на их вопросы. В частности, мы хотим вести постоянные разделы «Что мне не нравится и что мне хотелось бы иметь в языке MS BASIC» и «Обнаруженные ошибки». АПМБ будет пытаться установить контакты непосредственно с разработчиками языка фирмы Microsoft, возможно, их заинтересуют предложения пользователей России. Тем более, что ряд ошибок языка, замеченных нами, тянется из QuickBASIC в VisualBASIC.

И, конечно, будет представлена оперативная информация и рецензии о новинках программных средств, публикациях в печати, литературе, документации и так далее.

«Почему разговор идет только о Microsoft BASIC? А другие версии языка?» — эти вопросы нам уже задавали неоднократно.

Все очень просто. Обратите внимание, что среди авторов книг и статей по языкам, особенно по их сравнительному анализу, почти нет собственно программистов, в основном, это преподаватели вузов. А наш ИКЦ АПМБ был создан группой программистов-практиков, ведущих профессиональную разработку программных средств в области автоматизации научных исследований именно на языке Microsoft QuickBASIC 4.5. Под термином «профессиональная» мы понимаем то, что реализация разработок является главным источником нашего заработка. Поэтому мы просто не имеем возможности изучения других версий языков из чистого любопытства. Более того, мы отлично осознаем, что даже о семействе Microsoft BASIC нам известно далеко не все, например, о Visual BASIC for WINDOWS почти ничего.

Поэтому мы можем писать только о том, что знаем и с чем работаем. Но если кто-то захочет поделиться своими знаниями о других версиях языка: POWER, TRUE, BETTER и прочих, — то мы будем только рады и им всегда найдется место в журнале. Более того, мы уверены, что у нас есть общие темы и с пользователями других языков. Всю подготовку материалов этой рубрики будет выполнять дирекция ИКЦ АПМБ, поэтому со всеми предложениями и вопросами читатели могут обращаться непосредственно по адресу:

105058 Москва, а/я 4, ИКЦ АПМБ, А.А. Колесову
тел. (095) 366-25-02

Мы рассчитываем на Ваше, уважаемые читатели, участие в жизни новой рубрики «Монитора», по крайней мере, надеемся узнать Ваше мнение о нем.

Пишите, звоните, заходите в гости!

Ведущий рубрики, координатор ИКЦ АПМБ, Андрей Колесов

В жизни каждого программиста периодически наступают моменты, когда он непосредственно сталкивается с известным фактом — «любой компьютер имеет ограниченные ресурсы». И самая типичная ситуация — не хватает оперативной памяти. Еще 4 года назад, когда работали на ДВК с памятью 64 Кб, казалось, что 640 Кб — это безумно много и у нас-то проблем с этим не будет до пенсии. Но сегодня до пенсии еще далеко, а памяти уже недостаточно.

Проблема нехватки ресурсов имеет два основных решения:

1) замена компьютера на более мощный;

2) поиск и использование резервов за счет некоторой реконфигурации программы, а также оптимизации технологии ее разработки.

Естественно, первый вариант является наиболее радикальным решением и в перспективе от него никуда не уйти. Но сделать достаточно быстро это может далеко не каждый программист. Тем более, что часто уже при постановке задачи бывают жестко определены технические ограничения, например, минимальная конфигурация компьютера. Так что ресурсы приходится изыскивать.

Проблемы с оперативной памятью — общие соображения

Как известно, для работы программы в среде интерпретатора требуется загрузка всех необходимых процедур непосредственно в оперативную память. Поэтому для программиста, работающего в среде Microsoft BASIC, проблемы с оперативной памятью могут появиться еще на этапе разработки отдельного EXE модуля. И это является неизбежной расплатой за повышение скорости отладки программы с использованием интерпретатора.

На этом этапе разработки в системе QuickBASIC можно предложить такие варианты решения данной проблемы:

1) уменьшение размеров исходных текстов программы за счет сокращения выполняемых ею функций;

2) уменьшение размеров исходных текстов за счет оптимизации структуры программы (использование подпрограмм, процедур, конструкций $INCLUDE, разумное размещение процедур по исходным модулям и прочее);

3) работа в среде QB.EXE с использованием Quick библиотек.

Первые два варианта предполагают более или менее решительное изменение структуры самой программы, третий — чисто технологическую модернизацию работы в среде QB.EXE. У пользователей BASIC PDS 7.1 и Visual BASIC есть четвертый, более радикальный технический способ — использование расширенной памяти [1].

Если не принимать во внимание тривиального удаления из программы ее функций, как некоторых архитектурных излишеств, без которых прикладная система может нормально работать, первый вариант связан с реализацией исходной программы в виде двух или более исполняемых модулей. Безусловно, такая операция является довольно непростой задачей, но при создании больших комплексов от этого не уйти. Суть проблемы заключается в выделении достаточно автономных функциональных EXE модулей, передача управления между которыми может выполняться операторами CHAIN, RUN, SHELL, а обмен данными — с помощью общей области памяти, файлов, командных строк, кодов завершения программ и прочего.

Оптимизация исходных текстов программы подразумевает, прежде всего, выделение ее общих фрагментов в отдельные процедуры, INCLUDE файлы а также удаление из исходных модулей всего того, что не имеет непосредственного отношения к данной программе. При этом особое внимание следует уделить качеству используемых инструментальных программных модулей [2] и, в частности, оптимальному распределению процедур по исходным программным файлам. Необходимо помнить, что при использовании какой-либо процедуры в программу включается весь модуль, в котором она записана.

Если при разработке программы Вы используете загрузку всех процедур в виде исходных модулей, то наиболее простым решением проблемы с оперативной памятью является переход к использованию части процедур в форме Quick библиотек [2]. При работе с среде QB.EXE без библиотек можно загрузить не более 4500-5500 строк исходного текста, что соответствует исполняемому модулю размером 150-180 Кб (вариант Stand-Alone). При использовании Quick библиотек размеры программы могут быть существенно расширены: до 300-400(!) Кб. Опыт автора показывает, что при создании больших программ 50-80% их операторов приходится на инструментальные и вспомогательные процедуры, которые можно объединить в библиотеки.

Еще одной неожиданной, но легко решаемой проблемой в системе QuickBASIC является ограничение размеров отдельного исходного модуля — компилятор BC.EXE 4.5 не обрабатывает модули более 64 Кб (уже в версии 7.1 это ограничение снято). Но с этим сталкиваются обычно начинающие программисты, как правило, только что пересевшие с GW-BASIC. Разумеется, это самая простая проблема — исходный файл надо просто разделить на несколько модулей.

Второй круг проблем связан с нехваткой памяти для выполнения готового EXE модуля. Эта ситуация является общей для всех программистов, независимо от используемого языка. Для пользователя QuickBASIC она однозначно может появиться только при реализации многомодульных программных комплексов. Уже это предполагает, что программист должен обладать соответствующим опытом и квалификацией.

Здесь также видятся два основных решения:

- уменьшение размеров самих EXE модулей;
- структурные преобразования программного комплекса в целом.

Второй вариант связан опять-таки с принципиальной реконструкцией системы, к которой целесообразно приступать, когда другие резервы исчерпаны. Хотя и затягивать с ней не стоит, если ее перспектива становится очевидной.

Наиболее эффективным способом экономии оперативной памяти в QB 4.5 является использование EXE модулей, созданных в режиме «Requiring BRUN45.EXE» [2], [3]. В отличие от режима «Stand-Alone», каждый модуль получается меньше на

40-50 Кб, и поэтому, хотя требуется дозагрузка специального модуля поддержки (run-time module) BRUN45.EXE, экономия памяти появляется уже при загрузке двух программ одновременно (рис. 1.). За счет использования стандартного модуля BRUN45.EXE освобождается и дисковое пространство.

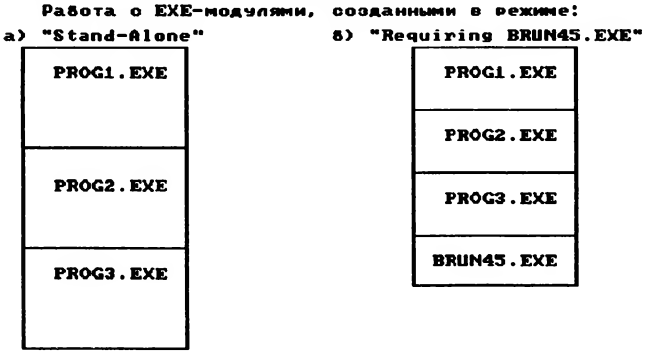


Рис. 1. Размещение в оперативной памяти исполняемых модулей при их последовательной загрузке (обращение «SHELL PROGn.EXE»)

Естественной расплатой за экономию памяти является некоторое увеличение времени на передачу управления от одного модуля к другому и обратно (рис. 2). При вызове программы PROG2 командой SHELL производится загрузка не одного, а двух файлов — самой программы и модуля поддержки. При ее завершении и возврате управления программе PROG1 происходит дозагрузка модуля поддержки.

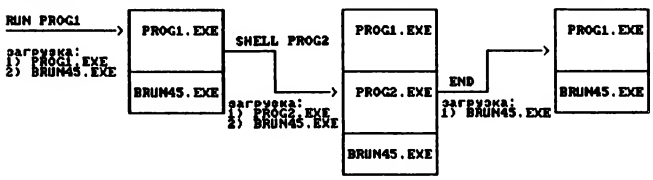


Рис. 2. Последовательность загрузки в оперативную память EXE модулей, созданных в режиме «Requiring BRUN45.EXE»

При передаче управление исполняемому файлу пользователя операционная система загружает BRUN45.EXE автоматически. Он должен находиться в текущем каталоге или каталоге, описанном в команде PATH. Если этот модуль не найден на диске, система выдает запрос:

```
Input run-time module path:
```

При работе в системе BASIC PDS 7.1 возможности программиста по решению проблем оперативной памяти на уровне EXE модулей гораздо шире: оптимизация программного кода, создание оверлейных структур, использование модулей поддержки пользователя и прочее [1], [4]. И, кроме того, использование расширенной памяти компьютера.

Каждый из перечисленных здесь вопросов достоин подробного обсуждения, но сейчас это не входит в задачу автора. Следует только заметить, что возможности улучшения качества программы и выбора оптимальной ее структуры во многом определяются подготовкой программиста. Поэтому квалифицированный специалист почти всегда сможет помочь начинающему программисту в определении наилучших вариантов модернизации его программ.

Сократить размеры EXE модуля можно на 20-30%

Поговорив о проблемах с оперативной памятью вообще, теперь рассмотрим некоторые практические рекомендации по минимизации размеров EXE модулей в системе Microsoft QuickBASIC. Суть вопроса заключается в следующем: соблюдая определенные правила программирования и управляя режимами компиляции программы, можно сократить размеры EXE модуля на 20-30%.

Поскольку мы будем вести разговор о технологии создания исполняемого модуля, следует напомнить о структуре BASIC программы [2]. С точки зрения формы программа состоит из одного или нескольких модулей, реализованных в виде отдельных исходных файлов, каждый из которых может обрабатываться автономно транслятором (компилятором или интерпретирующей системой). Каждый модуль состоит из логических программных блоков — главной части модуля (может быть пустым) и процедур: одной, нескольких или ни одной.

Для начала приведем информацию о размерах объектных файлов, полученных в результате компиляции двух исходных модулей с различными опциями (табл. 1). Модуль VIEWER.BAS содержит процедуры диалоговой программы, CURVEA.BAS — расчетной. Как видно, изменение размеров объектных, а значит, и исполняемых модулей в зависимости от режимов компиляции довольно значительное.

Работая в среде QuickBASIC, большинство программистов совершенно не обращают внимания на то, каким образом производится создание EXE модуля. Подвел курсор к позиции <RUN/MAKE EXE>, нажал Enter и жди получения исполняемой программы. Система сама определяет ключи компиляции без участия пользователя. Мы попробуем разобраться, каким образом она это делает и как программист может управлять этим процессом.

При создании EXE модуля пользователь в явном виде может управлять двумя режимами компиляции:

/D — включение средств диагностики (Produce Debug Code): проверка run-time ошибок и обработка аварийного завершения по CTRL+BREAK;

/O — компиляция в режиме «Stand-Alone», при отсутствии «ключа» — в режиме «Requiring BRUN45.EXE».

По умолчанию меню <MAKE EXE> предлагаются режимы без использования этих ключей. Наличие ключа /O не влияет на размеры объектного модуля, он просто определяет, какая библиотека должна подключаться при компоновке EXE модуля: BCOM45.LIB (/O) или BRUN45.LIB (без /O). В то же время из табл. 1 видно, что включение диагностики (/D) влияет на размеры довольно существенно. В этом нет ничего удивительного и тем более страшного — ключ /D нужно использовать только на этапе отладки программы.

В системе BASIC PDS 7.1 возможности выбора оптимальных режимов компиляции существенно расширены [4]: увеличен сам набор опций, и управление ими может выполняться непосредственно из среды

Таблица 1

Размеры объектных модулей при компиляции BC.EXE 4.5

	без ключей	/V	/W	/X	/E	/D	/V/X	/V/X/D
VIEWER.BAS								
байты	7841	10314	8312	10215	8236	12480	12283	16470
%	100%	132%	106%	130%	105%	159%	156%	210%
CURVEA.BAS								
байты	5534	7598	5642	7507	5637	7281	9119	10551
%	100%	137%	102%	135%	102%	132%	165%	190%

Таблица 2

Размеры объектных модулей при компиляции BC.EXE 4.5 и BC.EXE 7.1

	4.5	7.1					
	без ключей	без ключей	/G2	/Ot	/Fpa	/Fs	/S
VIEWER.BAS							
байты	7841	7644	7382	7644	8121	8004	7608
%	100%	97%	94%	97%	104%	102%	97%
CURVEA.BAS							
байты	5534	5334	5318	5536	7383	5404	5328
%	100%	96%	96%	100%	132%	98%	96%

QBX.EXE (рис. 3). В табл. 2 представлены сравнительные характеристики результатов трансляции двух исходных модулей компиляторами версий 4.5 и 7.1 с некоторыми ключами (описание ключей приведено на рис. 3). Следует подчеркнуть, что представленные данные не претендуют на полную характеристику влияния режимов компиляции на размеры объектных и исполняемых модулей. Тем более, что в BASIC PDS минимизация EXE модулей может в значительной степени осуществляться на уровне подключения библиотек соответствующей конфигурации.

Но вернемся к табл. 1 и попробуем разобраться с ключами /X/E и /V/W.

/X и /E — включение средств программной обработки ошибок

Ключ /X обеспечивает включение в объектный модуль средств, необходимых для работы конструкции «ON ERROR with RESUME {0 | NEXT | linelabel}», ключ /E — конструкции «ON ERROR with RESUME linelabel». То есть ключ /E является частным случаем по сравнению с /X, поэтому при компиляции режим /E/X равносилен /X. Тем не менее при создании выполняемого модуля QB.EXE автоматически формирует для каждого исходного файла ключи компиляции следующим образом:

/E — при наличии хотя бы одного оператора ON ERROR или RESUME linelabel в данном модуле;

/X — при наличии хотя бы одного оператора RESUME {0 | NEXT}.

Очевидно, что для обработки ошибок наиболее характерной является конструкция «ON ERROR with RESUME {0 | NEXT}» и, соответственно, компиляция в режиме /E/X, что приводит к увеличению объектного модуля на 30-35% за счет включения в него дополнительных средств контроля. Что можно сделать для предотвращения столь значительного роста объема EXE модуля?

На практике средства программной обработки ошибок, как правило, включаются для контроля лишь отдельных небольших фрагментов вычислительного процесса. Поэтому логично предложить следующее решение: локализовать фрагменты программы, для которых включается программная обработка ошибок,

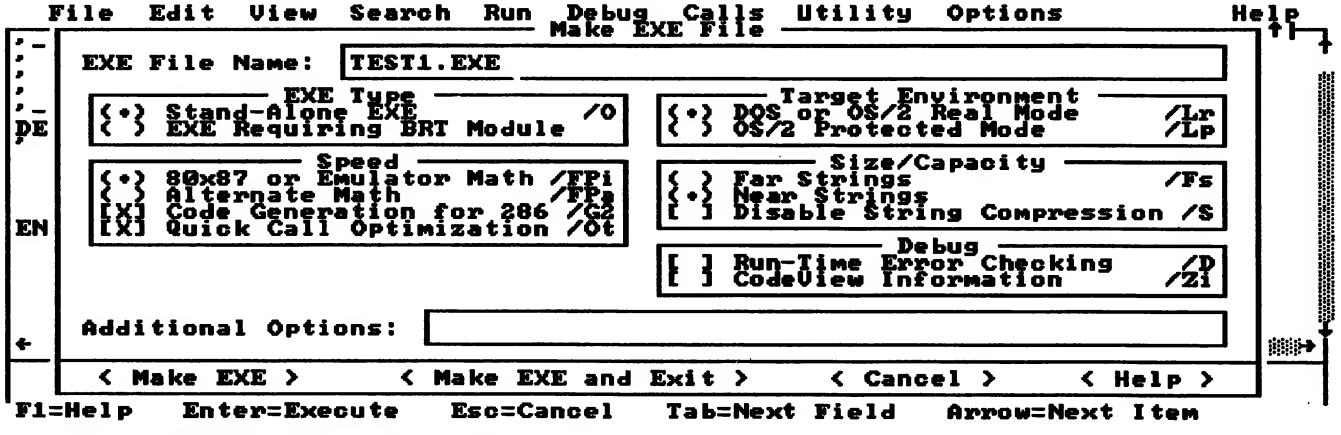


Рис. 3. Среда QBX (BASIC PDS 7.1). Окна создания исполняемого модуля

в виде небольших отдельных модулей. При этом никаких процедур, не связанных с этим режимом работы, в них включать не надо. Таким образом, компилироваться с ключами /E/X будет только небольшая часть программы, для которой это действительно необходимо. Проиллюстрируем это конкретным примером.

Довольно типичной ситуацией использования конструкции «ON ERROR with RESUME» является программный анализ наличия ошибки при открытии входного ASCII файла в режиме INPUT. Обычной ошибкой является обращение к несуществующему файлу (53 — File not found). Выделив соответствующий фрагмент программы в виде процедуры OpenInputFile и поместив ее в отдельный инструментальный модуль OPNIFILE.BAS (листинг 1), можно минимизировать размеры исполняемого модуля.

Следует также обратить внимание на то, что оператор ON ERROR инициализирует средства обработки ошибок для всех процедур, но только своего модуля. Поэтому, например (листинг 2), после инициализации обработки ошибок в модуле ERROR2.BAS (процедура OnErrorGoto) при обращении из любого места программы к процедуре открытия файла OpenInputFile2 будет проводиться программная обработка ошибки. Но при попытке открытия несуществующего файла в самой вызывающей программе (оператор с меткой M1:) ее выполнение будет прервано и выдана системная диагностика. Причем это будет происходить, даже если каким-то образом откомпилировать вызывающий модуль TEST2.BAS с ключом /X. ■

(Продолжение в следующем номере)

Литература

1. **А.А.Колесов, О.Р.Павлова. Все течет, все изменяется, или от «быстрого» Бейсика к «профессиональному»// Мир ПК.– 1992.– № 5.**
2. **А.А.Колесов, О.Р.Павлова. Профессиональная работа в среде Microsoft QuickBASIC.// Монитор.– 1993. — № 2, 3.**
3. **Microsoft QuickBASIC 4.5. — USA: Microsoft Corporation, 1987.**
4. **Microsoft BASIC 7.0 Professional Development System. — USA: Microsoft Corporation, 1989.**

Вопрос

Мне не удается передать свой тип данных (переменную, определенную с помощью оператора TYPE) в модули, используя команду COMMON SHARED.

Д.В.Дубин, Одесса

Ответ

В системе QuickBASIC, начиная с версии 4.0, кроме элементарных (Elementary Data Types) переменных — числовых, строковых — можно использовать тип данных, определяемых пользователем (User Defined Data Types). Это данные типа «запись», поэтому их часто называют RECORDS или STRUCTURES, с полями фиксированной длины. В версии 4.0 поля могли быть реализованы в виде элементарных переменных (разумеется, кроме строковых переменных переменной длины), кроме того, в версии 4.5 в качестве поля можно использовать другие записи, в BASIC PDS 7.0 — статические массивы.

Работа с данными типа «запись» обладает определенной спецификой по сравнению с традиционными элементарными типами данных. Это относится и к вопросу о передаче данных между модулями. Суть проблемы заключается в необходимости точного соответствия описания структуры переменной оператором «TYPE ... END TYPE» во всех модулях Вашей программы.

Ответ

При использовании команды COMMON SHARED это может выглядеть, например, так:

```
Модуль 1:                              Модуль 2:
TYPE Type1                             TYPE Type2
  i1 AS INTEGER                          i1 AS INTEGER
  a2 AS LONG                             a2 AS LONG
  s3 AS STRING*10                        s3 AS STRING*10
END TYPE                               END TYPE
COMMON SHARED Record1 as Type1         COMMON SHARED Record2 as Type2
```

В этом случае оба модуля будут работать с одной и той же переменной Record1/Record2, хотя, как видите, идентификаторы переменных и их описателей могут не совпадать (конечно, во избежание путаницы лучше использовать одинаковые имена). Главное — идентичное описание структуры, иначе правильная передача данных невозможна, вплоть до появления аварийных ситуаций. Аналогичная ситуация происходит с передачей элементарных переменных при обращении к модулям, загруженным в виде QLB-библиотеки (см. статью «Профессиональная работа с среде Microsoft BASIC», ч. 2, опубликованную в журнале «Монитор» 3.93). А здесь проблема заключается в том, что транслятор не может гарантировать автоматическую фиксацию несоответствия описаний в разных модулях и при загрузке всех процедур в виде исходных текстов, — ответственность ложится на программиста.

Особенно четко это видно при передаче данных в качестве параметров процедур:

```
Вызывающий модуль 1:                   Вызываемый модуль 2:
DECLARE SUB1 (Record AS ANY)
TYPE Type1 ... END TYPE                TYPE Type2 ... END TYPE
  ...
  DIM Record1 as Type1

CALL SUB1 (Record1)                    SUB SUB1 (Record2 AS Type2)
```

В описании параметров вызываемой процедуры DECLARE SUB1 записано ключевое слово ANY, которое подчеркивает, что используется некоторый тип данных, не известный транслятору.

Из всего сказанного можно сделать такие выводы:

1. Использование переменных типа «запись» требует повышенного внимания программиста, возникают проблемы соответствия их описания в разных модулях именно на этапе разработки программы, когда структура данных еще окончательно не определена. Некоторым ее решением может быть вынесение описаний в общий файл, текст которого включается в каждый модуль с помощь метакоманды $INCLUDE.
2. В своих программах мы практически не используем переменные типа «запись» как средство хранения и обработки данных. Причина этого — не только в проблемах, затронутых ранее, а в том, что они не обеспечивают динамическое описание структуры записи: оператор TYPE формирует ее жесткую структуру на уровне кода программы. А большинство практических приложений, особенно при работе с базами данных, требует, чтобы программа могла сама настраиваться на структуру данных. Для этого описатель переменной должен передаваться в вызываемый модуль вместе с ее содержимым (примерно так же, как передается описатель массива).

Пожелание реализации такой структуры данных можно адресовать разработчикам языка BASIC. А пока они размышляют над этим предложением, мы решили проблемы структуризации и динамического описания данных средствами BASIC в своем инструментальном комплексе QB-WDET «Оконный интерфейс работы с числовыми и структурированными данными».

3. Опытным программистам хорошо известно коварство передачи данных с помощью оператора COMMON, большинство из них почти забыли о нем. Единственный действительно необходимый случай, известный мне, — это передача данных между EXE модулями с помощью оператора CHAIN.

Несмотря на внешнюю простоту передачи данных между модулями с помощью COMMON, при создании довольно быстро возникают проблемы из-за несоответствия описаний переменных даже для не очень больших программ . Здесь полная аналогия с записями, но уже для любых типов данных.

Вопросы передачи данных между модулями непосредственно связаны с общей проблемой стиля программирования. И лучше всего она освещена в классической работе: Йодан Э. Структурное проектирование и конструирование программ: Пер. с англ.– М.: Мир, 1979.

А. Колесов

Листинг 1. Использование средств программной обработки ошибок

TEST1.BAS — пример рабочего фрагмента программы,

OPNIFILE.BAS — инструментальный модуль:
процедура открытия файла в режиме INPUT

```
DECLARE SUB OpenInputFile (File$, Nf%, kz%)
'-------------------------------------------------
'          Модуль TEST1.BAS
'        использование средств
'    программной обработки ошибок —
'    открытие файла в режиме "INPUT"
'-------------------------------------------------
    '...
    ' обращение к внешней процедуре
    File$ = "TEST1.TMP"
    CALL OpenInputFile(File$, Nf%, kz%)
    IF kz% > 0 THEN PRINT "Файл " + File$ + " не открыт"
    '...
END

'************************************************
'*************** Kolesov QB TOOLS*************** *
'*              QB-SERVICE_FILEWORK              *
'************************************************
'*                Модуль OPNIFILE.BAS            *
'*             процедура открытия INPUT файла    *
'************************************************
    DIM SHARED Kodz%
END
            ' обработка ошибок
ErrFile:    Kodz% = ERR: RESUME NEXT

SUB OpenInputFile (File$, Nf%, kz%)
            ' открытие Input файла
' ----------------------------------
' Вход:     File$ — имя файла
' Выход:    файл открыт   – Nf — номер файла, kz=0
'           ошибка        – Nf = 0, kz - код ошибки
''''''''''''''''''''''''''''''''''''''''''''''''''
    ON ERROR GOTO ErrFile  ' установка обработки ошибки
    Kodz% = 0              ' очистка флага ошибки
    Nf% = FREEFILE: OPEN "i", #Nf%, File$
    ON ERROR GOTO 0        ' сброс обработки ошибки
    kz% = Kodz%: IF Kodz% > 0 THEN Nf% = 0
                           ' анализ ошибки
END SUB
```

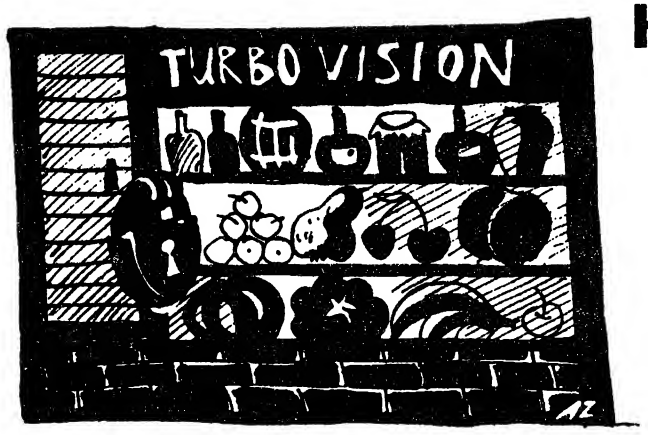
Листинг 2. Использование средств программной обработки ошибок

TEST2.BAS — пример рабочего фрагмента программы,

ONERROR2.BAS — процедуры, использующие
средства обработки ошибок

```
DECLARE SUB OnErrorGoto ()
DECLARE SUB OpenInputFile2 (File$, Nf%, kz%)
'-------------------------------------------------
'         Модуль TEST2.BAS
'     использование средств
' программной обработки ошибок
'-------------------------------------------------
    CALL OnErrorGoto     ' инициализация обработки ошибок
    '...
    ' открытие файла:
    CALL OpenInputFile2("FILE20.TMP", Nf1%, kz%)
    IF kz% > 0 THEN PRINT "Файл не открыт"
    '...
    Nf2% = FREEFILE
    ' при отсутствии файла FILE21.TMP здесь
    ' может произойти аварийное завершение программы:
M1:OPEN "i", #Nf2%, "FILE21.TMP"
    '...
END

'************************************************
'*       Модуль ONERROR2.BAS                    *
'*----------------------------------------------*
'* Программная обработка ошибок при окрытии     *
'* INPUT файла                                  *
'*   Процедуры:                                 *
'* OnErrorGoto — инициализация обработки ошибок *
'* OpenInputFile2 — открытие файла              *
'************************************************
    DIM SHARED Kodz%
END
    ' обработка ошибок
ErrFile:Kodz% = ERR: RESUME NEXT

SUB OnErrorGoto
    ON ERROR GOTO ErrFile  ' установка обработки ошибки
END SUB

SUB OpenInputFile2 (File$, Nf%, kz%)
    ' открытие Input файла
    ' (предварительно должна быть выполнена
    ' инициализация системы обработки ошибок
    ' процедурой OnErrorGoto)
'-------------------------------------------------------
' Обращение: см. OpenInputFile (OPNIFILE.BAS)
'''''''''''''''''''''''''''''''''''''''''''''''''''''''
    Kodz% = 0              ' очистка флага ошибки
    Nf% = FREEFILE: OPEN "i", #Nf%, File$
    kz% = Kodz%: IF Kodz% > 0 THEN Nf% = 0
                           ' анализ ошибки
END SUB
```

Программируем СУБД на Turbo Vision



С. Никоненко

...И напряженно, неустанно выжидали, когда же, наконец, представится случай вселить ужас и восхищение в чью-то простую душу.

Джером Д. Сэлинджер. Человек, который смеялся

Часть 3

Виртуальная база данных Введение

Основной идеей Data Vision, ее краеугольным камнем, является виртуальная база данных, реализованная в типе объекта **TDefDBV**. Программный интерфейс методов объекта виртуальной БД соответствуст общим правилам объекта **TDefDB**. Это неудивительно, поскольку **TDefDBV** является прямым наследником абстрактного объекта БД **TDefDB**.

Под виртуальной базой данных, как правило, понимается такая БД, структура данных которой не имеет физического аналога. В общем случае виртуальная БД представляет собой таблицу, графы которой могут ассоциироваться с полями различных физических и виртуальных БД, а также являться результатом некоторых функциональных зависимостей. Более того, эти объекты могут использовать различные типы БД одновременно (dBase, Paradox, Clarion, BTrieve — в одной таблице)! Это напоминает объектно-компонентную архитектуру Borland (BOCA), вернее, ее компоненты — так называемую оболочку Object Layer и открытый интерфейс к базам данных ODAPI, обеспечивающие «прозрачную» работу с файлами dBase и Paradox, а также другими БД посредством драйверов БД. Роль драйверов БД в Data Vision выполняют интерфейсные объекты (**TDefDBF** для dBase, **TDefPX** для Paradox, **TDefCla** для Clarion и так далее). Виртуальная БД, благодаря общему предку иерархии объектов БД и унифицированному интерфейсу виртуальных методов этого объекта, может соединять данные из файлов БД, с которыми связаны экземпляры различных интерфейсных объектов и других виртуальных БД. Такой объект называется неоднородной виртуальной БД. Однородная виртуальная БД может содержать поля из файлов БД, с которыми связаны экземпляры интерфейсных объектов одного типа (например, только dBase). Использование объекта виртуальной БД ничем не отличается от использования прочих объектов БД с точки зрения программиста.

В Data Vision введено также понятие фиктивной графы. Это графа, которая вообще не имеет физического аналога. Значение, отображаемое фиктивной графой, является результатом вычисления функциональных выражений.

Область применения виртуальных БД

Практически не существуст таких структур данных, при проектировании которых не использовались бы виртуальные БД. Это связано со следующим принципиальным моментом: механизм реализации реляций между различными БД выведен из программного интерфейса абстрактного предка БД — объекта **TDefDB** и интерфейсных объектов.

Только объект виртуальной базы данных и его наследники имеют средства для организации отношений — метод **TERelation**. Именно это позволяет устанавливать отношения между БД различной физической организации. Естественно, везде, где необходимы реляции, в Data Vision применяются объекты виртуальной БД.

Пример однородной виртуальной БД (SIMPLEV1.PAS)

В примерах программ, приведенных далее, рассматриваются различные варианты парамет-

ров конструктора объекта виртуальной БД. На основе примера 3 будет рассмотрено большинство ключевых моментов проектирования объекта виртуальной БД.

Итак, в примере 3 виртуальная БД составлена из двух БД в формате dBase. Первый файл называется **APP.DBF** и имеет 2 поля: поле наименования программного продукта **NAME** и поле кода группы **GCODE** программного продукта, соответствующего аналогичному полю из файла БД **GROUP.DBF**.

Содержимое **APP.DBF** приведено в табл. 4.

Таблица 4

База данных APP.DBF

NAME	GCODE
Borland Pascal	1
Quattro Pro	2
Microsoft C	1
Lotus 1-2-3	2
Microsoft Excel	2
Paradox	3
Microsoft Word	4
CA-Clipper	3
Zortech C/C++	1
WordPerfect	4

Файл **GROUP.DBF** также имеет два поля: уже упоминавшийся код группы **GCODE** и имя группы **GNAME**.

Содержимое **GROUP.DBF** приведено в табл. 5.

Таблица 5

Содержимое базы GROUP.DBF

GCODE	GNAME
1	Компилятор
2	Электронная таблица
3	СУБД
4	Текстовый процессор

Виртуальная БД также состоит из 2 полей: поля наименования программного продукта из **APP.DBF** и поля имени группы из **GNAME.DBF**. Инициализация такого объекта выглядит достаточно просто:

```
NewDB:=New(pOurDBV,Init(VirtID,
    'Виртуальная БД',
    NewField(AppID,DBP(AppID)^.TEFldNo('NAME'),
    New Field(GroupID,DBP(GroupID)^.TEFldNo('GNAME'),
    NIL))));
```

Естественно, перед этим необходимо убедиться в существовании объектов БД, соответствующих идентификаторам **AppID** и **GroupID**. Напомню, что функция Data Vision **DBP** возвращает указатель на объект, идентификатор которого передается ей в качестве параметра. Метод **TEFldNo** возвращает номер поля по имени поля, которое передается методу в качестве параметра. Функция **NewField** формирует список полей виртуальной БД. Ее первый параметр — идентификатор БД (**AppID**, **GroupID**), второй — номер поля указанной БД (здесь получается с помощью метода **TEFldNo**), а третий — ссылка на следующее поле.

Кроме этого, конструктору передается уникальный идентификатор виртуальной БД **VirtID**. Обратим внимание и на то, что инициализируется экземпляр типа **TOurDBV**. Возникают резонные вопросы: для чего порождать новый объект, почему нельзя обойтись объектом **TDefDBV**, и всегда ли это необходимо. Ответ такой: как правило, Вам придется это делать, поскольку нужно перекрыть метод **TERelation**. В примере 3 тело метода выглядит следующим образом.

```
function  TOurDBV.TERelation(N : Byte; Var Info : Byte) :
          Boolean;
var
    TmpInt : Integer;
begin
    if N<>2 then Exit;
    With Main^ do TmpInt:=TEGetInt(TEFldNo('GCODE'));
    if Item(2)^.Base^.TELocate(0,'I',TmpInt) then
          ExitState:=esSuccess
    else  ExitState:=esError+esNoRelation;
    TERelation:=ExitState=esSuccess
end;
```

Метод **TERelation** вызывается из методов доступа к данным **TEReplace** и **TEGetField**. Для того чтобы написать такой метод, надо обладать знаниями о его параметрах и о роли в объекте виртуальной БД поля **Main** и метода **Item**. Параметр N указывает номер поля, по которому вызывается метод. При чтении информации из поля (**TEGetField** и статические **TEGetXXXX**) **Amode** принимает значение константы Data Vision **riGetField**. При записи информации в поле (**TEReplace** и статические **TEPutXXXX**) **AMode** равно константе **riPutField**. Метод может возвращать в **AMode** значение константы **riProcessed**. Это означает, что при выполнении **TERelation** запись в поле уже произведена, и таким образом можно сэкономить время на выполнении тела метода **TEReplace**. Так работает Data Vision, но, разумеется, Вы можете расширить набор констант и режимов метода **TERelation**.

Объект **TDefDBV** и его наследники (наш объект **TOurDBV**) имеют поле ссылки на главную базу данных виртуальной БД — поле **Main**. По умолчанию этому полю присваивается ссылка на базу данных, идентификатор которой указан в первой функции **NewField** списка конструктора (в примере — **AppID**). Естественно, можно напрямую присвоить значение полю **Main**. Все навигации по виртуальной БД проводятся по главной БД, а отношения между компонентами виртуальной БД устанавливаются при чтении-записи полей (методы **TEGetField**/**TEReplace**). По умолчанию метод **TDefDBV.TERelation** устанавливает компоненты виртуальной БД на записи, номера которых равны текущему номеру записи главной БД. Именно поэтому необходимо перекрыть **TERelation** в нашем объекте.

Для доступа к базам-компонентам виртуальной БД используется функция **Item** с параметром — номером поля виртуальной БД, соответствующим порядку следования в списке **NewField** конструктора. Метод **Item** возвращает адрес структуры типа **TVirtualItem** Data Vision. Детально эта структура рассматривается в документации по Data Vision, однако для нас сейчас существенно одно из полей этой структуры — поле **Base**, содержащее ссылку на объект БД, обслуживающий данное поле виртуальной БД. После всех комментариев вернемся к нашему методу **TOurDBV.TERelation**. Отношение между БД программных продуктов и групп программных продуктов устанавливается по ключу **GCODE**. В главной БД с

ID=AppID значение кода считывается из второго поля и используется в качестве ключа при поиске в БД с ID=GroupID. Для чтения в формате целого числа используется статический метод TEGetInt (вызывающий внутри себя TEGetField). Для поиска по ключу используется виртуальный метод TELocate. Мы впервые встречаемся с этим методом, поэтому стоит остановиться на нем подробнее. Первый параметр указывает на поле поиска, однако если необходим поиск по ключу, то первый параметр должен равняться нулю, как в нашем случае. Второй параметр описывает формат буферной переменной — третьего параметра. В нашем случае это целочисленный ключ поиска. (Напомню, что в качестве файлов БД мы используем БД в формате dBase, но у других интерфейсных объектов значение первого параметра может интерпретироваться иначе.) Функция TELocate в случае нахождения заданного выражения позиционирует указатель текущей записи объекта на найденную запись файла БД и возвращает значение True, анализ которого позволяет установить поле ExitState,чье значение может возвратить метод TEExitState. Поле показывает результат выполнения последней операции над базой данных, используя набор констант Data Vision esXXXX.

Если поиск завершен успешно, ExitState устанавливается в код успешного завершения esSuccess, иначе устанавливается код ошибки esError и код конкретной ошибки (в примере ошибка реляции — esNoRelation). Сам метод TERelation является функцией, возвращающей True в случае успешной реляции и False в противном случае.

Вот, пожалуй, и все. Очевидно, что установка отношений между объектами БД — нетривиальный процесс, однако и сложного тут мало: нужно знать, как перекрыть метод TERelation. Все остальное просто. В результате всех наших усилий программа примера 3 выводит на дисплей содержимое виртуальной БД, и вот что мы увидим (табл. 6).

Таблица 6

Содержимое виртуальной базы данных

Borland Pascal	Компилятор
Quattro Pro	Электронная таблица
Microsoft C	Компилятор
Lotus 1-2-3	Электронная таблица
Microsoft Excel	Электронная таблица
Paradox	СУБД
Microsoft Word	Текстовый процессор
CA-Clipper	СУБД
Zortech C/C++	Компилятор
WordPerfect	Текстовый процессор

Пример неоднородной виртуальной БД (SIMPLEV2.PAS)

Неоднородная виртуальная БД не отличается от однородной практически ничем, за одним исключением. В нашем примере SIMPLEV2.PAS одним из компонентов виртуальной БД, а именно главной БД является база данных в формате Paradox (на ее месте может быть любая БД, формат которой интегрирован в Data Vision), в которой содержится та же информация, что и в аналогичном dBase файле предыдущего примера.

Если не конкретизировать параметры конструктора файла БД Paradox, то тексты примеров SIMPLEV1.PAS и SIMPLEV2.PAS ничем не отличаются. Поэтому я решил немного усовершенствовать пример 4: Paradox файл проиндексирован по полю наименования программного продукта (первичный индекс, включающий первое поле). Кроме этого, если реляция окончена неудачно (ExitState=esNoRelation+esError), в поле типа программного продукта выводится строка *'Неизвестный тип!'*, если из него читается строка, как в нашем случае. Для реализации этого перекрыт метод чтения поля TEGetField виртуальной БД.

```
procedureTOurDBV.TEGetField(N : Byte; FType : Char;
        Var AVal);
begin
   TDefDBV.TEGetField(N,FType,AVal);
   if ((ExitState and esNoRelation)=esNoRelation) and
   (N=2) and (UpCase(FType)='C') then
      String(AVal):='Неизвестный тип!'
end;
```

А теперь посмотрим на результат программы (табл. 7).

Таблица 7

Содержимое виртуальной базы данных

Borland Pascal	Компилятор
CA-Clipper	СУБД
Data Vision	Неизвестный тип!
Lotus 1-2-3	Электронная таблица
Microsoft C	Компилятор
Microsoft Excel	Электронная таблица
Microsoft Word	Текстовый процессор
Paradox	СУБД
Quattro Pro	Электронная таблица
WordPerfect	Текстовый процессор
Zortech C/C++	Компилятор

Результат налицо: записи виртуальной БД отсортированы по алфавиту, соответственно индексу главной БД. В Paradox файл APP.DB добавлена запись с полем NAME = Data Vision и с кодом GCODE = 5. Такой код отсутствует в GROUP.DBF, и поэтому мы получаем неизвестный тип программного продукта.

Пример фиктивной графы (SIMPLEV3.PAS)

В примере 5 в виртуальную БД введена фиктивная графа. Содержимое этого поля в примере довольно бессмысленно, однако хорошо иллюстрирует сам подход.

Для организации фиктивной графы необходимо в конструкторе указать для соответствующей функции NewField нуль вместо идентификатора базы данных.

```
NewDb := New(pOurDbv,Init(VirtID,
   'Виртуальна я БД',
   NewField(AppID,DBP(AppID)^.TEFIdNO('NAME'),
   NewField(GroupID,DBP(GroupID)^.TEFIdNO('GNAME'),
   New Field(0,0,
   NIL))));
```

Функциональное выражение для фиктивной графы задается в методе TEGetField. Это сумма кодов сим-

волов, составляющих название программного продукта. Метод TEGetField возвращает корректное значение для фиктивной графы при любом формате чтения из нее. Дополнительно перекрываются:

а) метод записи TEReplace, чтобы запретить запись в поле фиктивной графы;

б) метод, возвращающий информацию о длине поля TEGetLen;

в) метод, возвращающий информацию о формате поля TEGetFldForm.

Надо отметить, что для организации фиктивной графы, если она последняя у БД, необязательно перекрывать объект виртуальной БД TDefDBV; достаточно перекрыть интерфейсный объект. Однако наследование от объекта виртуальной БД более корректно, и вдобавок располагает специальными средствами, поддерживающими механизм фиктивных граф. Результат выполнения программы примера 5 представлен в табл. 8. ■

Таблица 8

Содержимое виртуальной базы данных

Borland Pascal	Компилятор	1334
CA-Clipper	СУБД	896
Data Vision	Неизвестный тип!	1042
Lotus 1-2-3	Электронная таблица	807
Microsoft C	Компилятор	1049
Microsoft Excel	Электронная таблица	1479
Microsoft Word	Текстовый процессор	1394
Paradox	СУБД	719
Quattro Pro	Электронная таблица	1089
WordPerfect	Текстовый процессор	1125
Zortech C/C++	Компилятор	1034

Литература

1. Ли Мантелман. Объектно-ориентированные СУБД зовут пользователя к переменам// Мир ПК.– 1992.– № 5.
2. Архитектура BOCA// БорИС.– 1993.– № 8.

INTERLINK

сети

100 Mb/s

COAX
TWISTED-PAIR
FIBER-OPTIC

FDDI
ARCNET
ETHERNET

Оборудование из США по Вашему заказу- - за 15 дней

Streamer, UPS, Scanner, Plotter, Digitizer.Продаем и устанавливаем сетевое оборудование.Поставляем программное обеспечение NOVELL.Разрабатываем топологию локальной сети.Возможна установка техники у заказчика в пределах России.

205-6410, 205-6411, 205-6920, 205-6988, fax (095) 956-70-21.
Москва, Глубокий пер. д. 5/11

SIREX 486 ISA/EISA
УСТАНОВКА СЕТЕЙ

Монитор–Аспект — 1993

Подписной купон

3

Монитор Аспект

Адрес _____

Ф.И.О. или название организации _____

Стоимость подписки на 1 номер журнала «Монитор–Аспект» для России — 300 рублей, для стран СНГ —600 руб. В 1993 году будет выпущен еще 1 номер. Перечислите сумму оплаты на счет НТО "Софт-Москва": р/с 467002 в КБ "Банк развития XXI век", к/с 161816 в РКЦ ГУ ЦБ РФ по г. Москве, МФО 201791.

Вышлите подписной купон и квитанцию об оплате по адресу: 117419 Москва, а/я 765.

Пример 3

```
{ ---------------------------------------------------------------------------

                    ОДНОРОДНАЯ ВИРТУАЛЬНАЯ БД

                       TURBO PASCAL v 6.0-7.0
                                ***
                  CopyRight (C) 1993, Никоненко С.Б.
                  Лаборатория программных средств SHIFT

                            SIMPLEV1.PAS

--------------------------------------------------------------------------- }

Uses Dos,
     Objects, Drivers,
     DVDefine,
     DVDefDB,
     DVdBase;              { Модуль DVision DBF Interface }

Const
     AppID    = 1;        { ID APP.DBF }
     GroupID  = 2;        { ID GROUP.DBF }
     VirtID   = 3;        { ID виртуальной БД }

Type
     pOurDBV = ^TOurDBV;                          { объект виртуальной БД }
     TOurDBV = object (TDefDBV)
          function TERelation(N : Byte; Var Info : Byte) : Boolean; virtual;
end;

Var
     NewDB    : pDefDB;                           { Ссылка на объект }
     RecInfo  : record                            { Запись для FormatStr }
          App       : PString;
          Group     : PString;
     end;
     TmpStr   : String;

                          (*** TOurDBV ***)

function TOurDBV.TERelation(N : Byte; Var Info : Byte): Boolean;
var
     TmpInt : Integer;
begin
     if N < > 2 then Exit;                        { Выход, если не 2 поле }
     { Считали код группы программных продуктов }
     With Main^ do TmpInt: = TEGetInt(TEFldNo('GCODE'));
                                                  { Поиск в справочнике групп }
     if   Item(2)^.Base^.TELocate(0,'I',TmpInt) then
          ExitState: = esSuccess
     else ExitState: = esError + esNoRelation;
     TERelation : =  ExitState = esSuccess
end;

  (* Процедура аварийного останова с предварительным сообщением
                          причины ошибки *)

procedure HaltError(Mess : String);
begin
     Writeln(Mess);
     Halt(1)
end;

begin
     NewDB: = New(pDefDBF,Init(AppID,             { ID БД APP.DBF }
                 'APP.DBF',                       { Имя БД }
                 '',                              { Нет индекса }
                 0,
                 NoUnique,
                 False));
     If NewDB = nil then HaltError('Не могу открыть APP.DBF');
     NewDB: = New(pDefDBF,Init(GroupID,           { ID БД GHROUP.DBF }
                 'GROUP.DBF',
                 'GROUP.NDX',                     { Имя индекса }
                 1,                               { Ключевое поле }
                 NoUnique,                        { Неуникальный ключ }
                 False));                         { Без автоиндексации }
     if NewDB = nil then HaltError('Не могу открыть GROUP.DBF');
     NewDB: = New(pOurDBV,Init(VirtID,            { ID виртуальной БД }
                 'Виртуальная БД',                { Имя ВБД }
                 NewField(AppID,DBP(AppID)^.TEFldNo('NAME'),
                 NewField(GroupID,DBP(GroupID)^.TEFldNo('GNAME'),
                 NIL))));
     if NewDB = nil then HaltError('Не могу открыть виртуальную БД');
     With DBP(VirtID)^,RecInfo do
     begin
          TEGoTop;
          Writeln('Содержимое виртуальной базы данных');
          Writeln;
          While not TEEOF do
          begin
               App: = NewStr(TEGetStr(1));
               Group: = NewStr(TEGetStr(2));
               FormatStr(TmpStr,'%-15S '#179' %-25S',RecInfo);
               Writeln(TmpStr);
               DisposeStr(App);
               DisposeStr(Group);
               TEGoNext
          end
     end;
     DoneDBList
end.
```

Пример 4

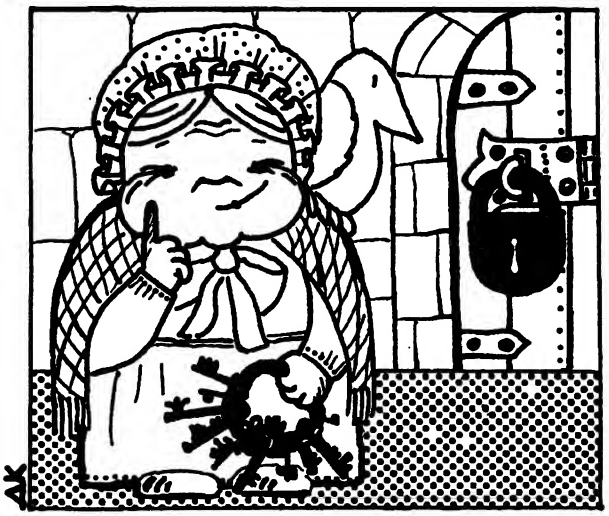
```
{ ---------------------------------------------------------------------------

                    НЕОДНОРОДНАЯ ВИРТУАЛЬНАЯ БД

                       TURBO PASCAL v 6.0-7.0
                                ***
                  CopyRight (C) 1993, Никоненко С.Б.
                  Лаборатория программных средств SHIFT

                            SIMPLEV2.PAS

--------------------------------------------------------------------------- }

Uses Dos,
     Objects,Drivers,
     DVDefine,
     DVDefDB,
     DVdBase,
     DVPX;

Const
     AppID    = 1;                                { ID БД APP.DB }
     GroupID  = 2;
     VirtID   = 3;

Type
     pOurDBV = ^TOurDBV;
     TOurDBV = object (TDefDBV)
          procedure TEGetField(N : Byte; FType : Char; Var AVal); virtual;
          function TERelation(N : Byte; Var Info: Byte) : Boolean; virtual;
     end;

Var
     NewDB    : pDefDB;
     RecInfo  : record
          App       : PString;
          Group     : PString;
     end;
     TmpStr   : String;

                          (*** TOurDBV ***)

procedure TOurDBV.TEGetField(N : Byte; FType : Char; Var AVal);
begin
     TDefDBV.TEGetField(N,FType,AVal);
                                  { Обработка ошибки связывания БД }
     if   ((ExitState and esNoRelation) = esNoRelation) and
          (N = 2) and (UpCase(FType) = 'C') then
               String(AVal): = 'Неизвестный тип!'
end;

function TOurDBV.TERelation(N : Byte; Var Info : Byte) : Boolean;
var
     TmpInt : Integer;
begin
     if N < > 2 then Exit;
     With Main^ do TmpInt: = TEGetInt(TEFldNo('GCODE'));
     if Item(2)^.Base^.TELocate(0,'I',TmpInt) then
          ExitState: = esSuccess
     else ExitState: = esError + esNoRelation;
     TERelation: = ExitState = esSuccess
end;

procedure HaltError(Mess : String);
begin
     Writeln(Mess);
     Halt(1)
end;

begin
     EnginePath: = '';     { Оверлей PXENGINE.OVL в текущем каталоге }
     InitParadox(@ParaDefault);
                  { Инициализация исполняющей системы PARADOX }
     NewDB: = New(pDefPX,Init(AppID,         { ID БД APP.DB }
                 'APP.DB',                   { Имя БД }
                 0,                          { Первичный индекс }
                 True,                       { Автосохранение }
                 1));                        { Количество полей индекса }
     if NewDB = nil then HaltError('Не могу открыть APP.DBF');
     NewDB: = New(pDefDBF,Init(GroupID,
                 'GROUP.DBF',
                 'GROUP.NDX',
                 1,
                 NoUnique,
                 False));
     if NewDB = nil then HaltError('Не могу открыть GROUP.DBF');
```

```
        NewDB: = New(pOurDBV,Init(VirtID,
                  'Виртуальная БД',
                  NewField(AppID,DBP(AppID)^.TEFldNo('NAME'),
                  NewField(GroupID,DBP(GroupID)^.TEFldNo('GNAME'),
                  NIL))));
        if NewDB = nil then HaltError('Не могу открыть виртуальную БД');
        With DBP(VirtID)^,RecInfo do
        begin
             TEGoTop;
             Writeln('Содержимое виртуальной базы данных');
             Writeln;
             While not TEEOF do
             begin
                  App: = NewStr(TEGetStr(1));
                  Group: = NewStr(TEGetStr(2));
                  FormatStr(TmpStr,'%-15S '#179' %-25S',RecInfo);
                  Writeln(TmpStr);
                  DisposeStr(App);
                  DisposeStr(Group);
                  TEGoNext
             end
        end;
        DoneDBList;
        ExitParadox   { Выход из исполняющей системы PARADOX }
    end.
```

Пример 5

```
{ ---------------------------------------------------------------------------

                        ФИКТИВНАЯ ГРАФА

                    TURBO PASCAL v 6.0-7.0
                            ***
                  CopyRight (C) 1993, Никоненко С.Б.
                Лаборатория программных средств SHIFT

                        SIMPLEV3.PAS

--------------------------------------------------------------------------- }

Uses Dos,
     Objects,Drivers,
     DVDefine,DVUtil,
     DVDefDB,
     DVdBase,
     DVPX;

Const
     AppID     = 1;
     GroupID   = 2;
     VirtID    = 3;

Type
     pOurDBV =  ^TOurDBV;
     TOurDBV =  object (TDefDBV)
          procedure TEGetField(N : Byte; FType : Char; Var AVal); virtual;
          procedure TEGetFldForm(N : Byte; Var FType : Char; Var Len,
                                 Dec : Byte); virtual;
          function TEGetLen(N : Byte) : Byte; virtual;
          function TERelation(N : Byte; Var Info : Byte) : Boolean; virtual;
          procedure TEReplace(N : Byte; FType : Char; Var AVal); virtual;
     end;

Var
     NewDB    : pDefDB;
     RecInfo  : record
          App       : PString;
          Group     : PString;
          Check     : PString;
     end;
     TmpStr    : String;

                        (*** TOurDBV ***)

procedure TOurDBV.TEGetField(N : Byte; FType : Char; Var AVal);
var
     TmpStr    : String;
     I         : Byte;
     Sum       : Integer;
begin
     case N of
     3 : begin
          TmpStr: = LTrim(RTrim(TEGetStr(1)));
          Sum: = 0;                    { Цикл подсчета контрольной суммы }
          for i: = 1 to Length(TmpStr) do Sum: = Sum + Ord(TmpStr[i]);
          case FType of                { Отработка форматов Data Vision }
          'I' : Integer(AVal): = Sum;                { Короткое целое }
          'I' : LongInt(AVal): = LongInt(Sum);       { Длинное целое }
          'C' : begin                                { Строка }
               Str(Sum:5,TmpStr);
               String(AVal): = TmpStr
          end;
          'D' : With TDateRec(AVal) do               { Дата }
          begin
               yy: = 0;
               mm: = 0;
               dd: = 0
          end;
          'R' : Real(AVal): = Sum*1.0;               { Вещественное }
          end;
     end;
     else TDefDBV.TEGetField(N,FType,AVal);
     end;
     if   ((ExitState and esNoRelation) = esNoRelation) and
          (N = 2) and (UpCase(FType) = 'C') then
               String(AVal): = 'Неизвестный тип!'
end;

procedure TOurDBV.TEGetFldForm(N : Byte; Var FType : Char;
                               Var Len, Dec : Byte);
begin
     if N = 3 then                         { Для фиктивного поля }
     begin
          FType: = 'C';                    { Символьный тип поля }
          Len: = 5;                        { Длина }
          Dec: = 0;                        { Десятичные разряды }
     end
     else TDefDBV.TEGetFldForm (N, FType, Len, Dec)
end;

function TOurDBV.TEGetLen(N : Byte) : Byte;
begin
     if N = 3 then TEGetLen: = 5
     else TEGetLen: = TDefDBV.TEGetLen(N)
end;

function TOurDBV.TERelation(N : Byte; Var Info : Byte) : Boolean;
var
     TmpInt: Integer;
begin
     if N < > 2 then Exit;
     With Main^ do TmpInt: = TEGetInt(TEFldNo('GCODE'));
     if Item(2)^.Base^.TELocate(0,'I',TmpInt) then
          ExitState: = esSuccess
     else  ExitState: = esError + esNoRelation;
     TERelation: = ExitState = esSuccess
end;

procedure TOurDBV.TEReplace(N : Byte; FType : Char; Var AVal);
begin
     if N < > 3 then TDefDBV.TEReplace(N,FType,AVal)
end;

procedure HaltError(Mess : String);
begin
     Writeln(Mess);
     Halt(1)
end;

begin
     EnginePath: = '';     { Оверлей PXENGINE.OVL в текущем каталоге }
     InitParadox(@ParaDefault);
                    { Инициализация исполняющей системы PARADOX }
     NewDB: = New(pDefPX,Init(AppID,
          'APP.DB',
          0,
          True,
          1));
     if NewDB = nil then HaltError('Не могу открыть APP.DBF');
     NewDB: = New(pDefDBF,Init(GroupID,
          'GROUP.DBF',
          'GROUP.NDX',
          1,
          NoUnique,
          False));
     if NewDB = nil then HaltError('Не могу открыть GROUP.DBF');
     NewDB: = New(pOurDBV,Init(VirtID,
          'Виртуальная БД',
          NewField(AppID,DBP(AppID)^.TEFldNo('NAME'),
          NewField(GroupID,DBP(GroupID)^.TEFldNo('GNAME'),
          NewField(0,0,
          NIL)))));
     if NewDB = nil then HaltError('Не могу открыть виртуальную БД');
     With DBP(VirtID)^,RecInfo do
     begin
          TEGoTop;
          Writeln('Содержимое виртуальной базы данных');
          Writeln;
          While not TEEOF do
          begin
               App: = NewStr(TEGetStr(1));
               Group: = NewStr(TEGetStr(2));
               Check: = NewStr(TEGetStr(3));
               FormatStr(TmpStr,'%-15S '#179' %-25S'#179'%-5S',RecInfo);
               Writeln(TmpStr);
               DisposeStr(App);
               DisposeStr(Group);
               TEGoNext
          end
     end;
     DoneDBList;
     ExitParadox   { Выход из исполняющей системы PARADOX }
end.
```

Коммерческие системы шифрования: основные алгоритмы и их реализация



В. Водолазкий

Часть 3. Системы с открытым ключом

Некоторое время назад в [1] мы начали разговор о системах шифрования с открытым ключом. Сегодня мы продолжим разговор и рассмотрим еще две коммерческие системы шифрования, не столь известные, как RSA. Заинтересованного читателя можно отослать сразу к первоисточнику [2].

Рюкзаковая система

Свое название эта система шифрования получила от одноименной задачи из области теории чисел (иногда ее называют задачей об упаковке ранца), которую мы попытаемся сейчас сформулировать. Итак,

Задача об упаковке рюкзака

Пусть задано множество целых чисел $A = \{a_1,a_2,\ldots,a_n\}$ и некоторое целое число C. Необходимо найти такое подмножество элементов A, чтобы их сумма равнялась C; иначе говоря, необходимо сформировать вектор $M = (m_1,m_2,\ldots,m_n)$, такой что

$$C = \sum_{i=1}^{n} a_i * m_i$$

Пусть $A = \{7,10,12,25,38,46\}$ и $C = 73$. В этом случае

$$M = (0,1,0,1,1,0)$$

$$C = 10 + 25 + 38$$

Поиск вектора M относится к очень сложному классу задач, называемых иногда «некорректно поставленными задачами» [2]. Тем не менее, существует некоторое подмножество этой задачи — «укладка простого рюкзака». В простом рюкзаке последовательность $\{a_1,a_2,\ldots,a_n\}$ является возрастающей, то есть для любого элемента множества выполняется соотношение

$$a_i > \sum_{j=1}^{i-1} a_j$$

для всех $i = 2\ldots n$.

Так, например, $A = \{1,3,8,14,29\}$ является простым рюкзаком.

В этой постановке задача нахождения вектора M решается гораздо проще. Для формирования вектора $M = (m_1,m_2,\ldots,m_n)$ нам достаточно использовать следующую процедуру (это легче написать на Pascal, чем рисовать алгоритмы):

```
CONST
   N = 5;          { длина вектора M и множества A }
TYPE
   Vect : array [1..N] of Integer;
CONST
   A : Vect = (1,3,8,14,29); { инициализируем
                               исходное множество }
Function KnapSack  (C : Integer;
                    Var M : Vect) : boolean;
Var
   I : Integer;                  { внутренний счетчик }
begin
```

```
    for I := 1 to N do M[I] := 0;  { инициализация массива }
    for I := N downto 1 do
        IF C >= A[I] then
        begin
            M[I] := 1;
            C := C — A[I]
        end;
    If C = 0    then KnapSack := true
                else KnapSack := false
end;
```

Функция KnapSack принимает значение TRUE, если для данного числа существует разложение по вектору A (в этом случае в M возвращается найденное разложение), и значение FALSE, если разложение найти не удалось. Так, например, если C окажется больше суммы всех элементов множества A, построить вектор M не удастся. И, вообще говоря, остается открытым вопрос, можно ли сформировать вектор M для заданного множества A для всех чисел, удовлетворяющих условию

$$0 < C < An$$

Конечно, такие вопросы легче задавать, чем на них отвечать. Автор честно признается, что сам ответа не знает и будет благодарен, если кто-либо из читателей его просветит.

Ответ, судя по всему, знали и два буржуазных специалиста — Мэркли и Хеллман, предложившие систему с открытым ключом, основанную на решении задачи об укладке простого рюкзака. Посмотрим, как это сделано у них.

Каждый из пользователей системы U:

1) выбирает неубывающую последовательность $A = \{a_1,a_2,\ldots,a_n\}$;

2) выбирает целое число u, удовлетворяющее условию

$$u > \sum_{i=1}^{n} a_i$$

3) выбирает целое число W такое, что наибольший общий делитель [3] U и W равен 1, а затем вычисляет W^{-1} — число, обратное W mod U.

Несколько отвлечемся и поясним, что же такое W^{-1}.

Это целое число, которое удовлетворяет условию:

$$W * W^{-1} \bmod U = 1$$

4) преобразует последовательность A в множество $B = \{b_1,b_2,\ldots,b_n\}$ по правилу:

$$b_i = W * a_i \bmod u$$

В результате объявляется открытый ключ B, а в секрете сохраняется A, W^{-1} и U.

Рассмотрим теперь, как в этой системе осуществляется шифрование и расшифровка сообщений.

Шифрование

Пусть Алекс ([1]) пытается послать N-битовое сообщение $M = (m_1,m_2,\ldots,m_n)$ Юстасу, объявившему открытый ключ B. Для этого Алекс вычисляет целое число C

$$C = \sum_{i=1}^{n} m_i * b_i$$

Полученное число C передается в канал связи.

Расшифровка

Для того чтобы прочитать сообщение, Юстас производит следующие операции:

$$W^{-1} * C = W^{-1} * \sum_{i=1}^{n} m_i * b_i = \sum_{i=1}^{n} m_i * a_i \bmod U$$

Здесь

$$\sum_{i=1}^{n} m_i * a_i < U$$

Получим

$$\sum_{i=1}^{n} m_i * a_i = W^{-1} * C \bmod U$$

Поскольку последовательность $A = (a_1,a_2,\ldots,a_n)$ возрастающая, то пользователь Юстас может применить приведенный алгоритм для восстановления сообщения M из

$$C' = W^{-1} * C \bmod U.$$

Поясним сказанное примером.

Предположим, мы имеем множество $A = \{1,4,6,13,27\}$.

Выберем теперь U = 59, W = 13. Тогда $W^{-1} = 50$ и, соответственно, $B = \{13,52,19,51,56\}$.

Теперь, для того чтобы послать сообщение M = (1 0 0 1 1), Алекс должен будет вычислить:

$$C = \sum_{i=1}^{5} m_i * b_i = 13 + 51 + 56 = 120$$

Чтобы прочитать принятое сообщение C = 120, Юстас должен прежде всего вычислить

$$C' = 120 * 50 \bmod U = 6000 \bmod 59 = 41$$

Используя быстрый алгоритм разложения числа по вектору M, мы получаем

$$\sum_{i=1}^{5} m_i * a_i = 41 = 1 + 13 + 27$$

Это соответствует M = (1 0 0 1 1).

Все это просто, красиво и элегантно вычисляется даже на 4-разрядных микропроцессорах. Но, к сожалению, в

1982 году был найден алгоритм, позволяющий «проломить» криптосистему Меркли-Хеллмана. Тем не менее, если передаваемая Вами по каналу связи информация стареет в течение нескольких суток, Вы можете с успехом пользоваться данной системой, не тратя вычислительные ресурсы на явно избыточные (по надежности) системы типа DES, RSA и так далее.

Система шифрования ElGamal

Если Вы читали предыдущую часть статьи, то, вероятно, помните, что задача разложения на сомножители больших целых чисел и задача вычисления дискретного логарифма сопоставимы между собой по трудоемкости решения. Но если система, основанная на разложении целых чисел, достаточно хорошо известна хотя бы понаслышке (RSA), то о системах с открытым ключом, основанных на вычислении дискретного логарифма, говорят почему-то меньше. А между тем еще в 1985 году Т. Эль Гамал опубликовал спецификации такой системы, которая продемонстрировала свою эффективность и до сих пор с успехом используется в коммерческих системах связи.

В этой системе всем пользователям сообщается большое (несколько сотен десятичных разрядов) целое число, а также число W, такое что

$$0 < W < P-1$$
$$W^{P-1} = 1 \pmod P$$

Но при этом

$$W^d \neq 1 \pmod P$$

для всех $1 < d < P-1$.

Пусть теперь многострадальный Алекс станет пользователем системы ElGamal.

Частный (он же секретный, он же приватный) ключ Алекса представляет собой целое число X_a, выбираемое произвольным образом в диапазоне:

$$1 \le X_a \le P-1$$

А открытый ключ Алекса — это целое число

$$Y_a = W^{X_a} \pmod P$$

Теперь предположим, что не менее многострадальный Юстас хочет послать Алексу сообщение M, которое представляет собой целое число в диапазоне

$$1 \le M \le P-1$$

Рассмотрим, как будет выполняться шифрование и расшифровка этого сообщения в системе ElGamal.

Шифрование

Процедура шифрования осуществляется в три приема.

1. Юстас выбирает случайное число K, такое что

$$1 \le K \le P-1$$

2. После этого Юстас вычисляет

$$R = Y_a^K \bmod p$$

3. И, наконец, Юстас производит шифрование сообщения M в пару элементов:

$$(C_1,C_2) = (W^K, M * R) = (W^K, M * W^{X_a * K})$$

Эта пара элементов и передается в канал связи.

Расшифровка

Получив пару (C_1,C_2), Алекс производит следующие действия.

1. Восстанавливает число R по правилу:

$$R = Y_a^K = W^{X_a * K} = (W^K)^{X_a} = C_1^{X_a} \pmod P$$

2. Восстанавливает M путем деления C_2 на R.

Интересно, что в результате читательских откликов на предыдущие публикации, в частности, [1], мы получили очень интересные программы вычисления модульной экспоненты, которые будут опубликованы в ближайших номерах нашего журнала. Поэтому автор не рискнет еще раз «позориться» с программой на Васике* и в очередной раз предоставляет читателям возможность отличиться. Мы готовы опубликовать эффективные алгоритмы шифрования и расшифровки данных в описываемых нами системах шифрования.

Электронная подпись

Предположим, Вы являетесь пользователем системы передачи данных с криптографическим закрытием информации. И вот на Ваше приемное устройство поступает Важное сообщение. Какова будет Ваша реакция, если от поступившей информации зависит, в какую графу — доходов или расходов — придется списывать, например, 2 млн. руб. (или, если предпочитаете мыслить более масштабно, долларов). Прежде чем предпринимать какие-либо действия, Вы должны быть уверены в следующих фактах:

- сообщение отправлено Вам вполне определенным лицом (тем, кто его подписал);
- во время передачи сообщения не произошло никаких искажений текста (наряду с происками

* Как выяснил когда-то корреспондент журнала «Микропроцессорные средства и системы», Васик — это универсальный язык программирования, разработанный специально для ПЭВМ «Ириша». Однако коварные шпионы из империалистического далёка похитили его и начали массово внедрять у себя. В Штатах наше ласковое название было искажено на Basic. Вот такая жуткая история! (Прим. автора)

старины Мюллера возможно просто искажение данных при передаче через канал).

Для решения этих проблем и используются системы электронной подписи, которые, несмотря на кажущуюся сложность, вполне по силам разработчикам среднего уровня квалификации. Как правило, подобные средства наиболее важны для систем с открытым ключом, каковые мы и возьмем в качестве основы для рассмотрения.

Еще раз вернемся к Алексу и Юстасу (старина Мюллер нам в этом случае не понадобится) [1], [4], [5]. Предположим, что Алекс и Юстас являются пользователями системы с открытым ключом и желают передать сообщение в тот или иной конец. Как Алекс может удостовериться, что на противоположном конце линии находится именно Юстас и наоборот? Для этого используется следующий протокол обмена:

1. Юстас выбирает случайное число X, зашифровывает его открытым ключом Алекса и передает Алексу

$$Y = E_a(X)$$

2. Алекс, используя свой секретный ключ D_a для расшифровки сообщения, передает $D_a(Y)$ обратно Юстасу.

3. Если Юстас, проверив принятый код, обнаруживает, что он получил число X, то считает, что на противоположном конце линии действительно находится Алекс.

Приведенная схема аутентификации пользователей, как уже указывалось, предназначена для систем с открытым ключом. Однако и для более распространенных сегодня систем, подобных DES — традиционным криптографическим системам с секретным ключом — можно организовать похожую схему.

Наша задача состоит в том, чтобы сгенерировать аутентификатор, используемый алгоритмом DES в режиме CBC (cipher block chaining). Исходный текст разделяется в соответствии с алгоритмом на блоки по 64 бита каждый. Первый блок исходного текста складывается по модулю два (операция XOR) с начальным вектором, затем зашифровывается с помощью секретного ключа, известного как отправителю, так и получателю сообщения. Полученный 64-битовый блок, в свою очередь, складывается по модулю два со вторым блоком исходного текста и снова шифруется. Мы получаем второй 64-битовый блок, и операция повторяется до окончания исходного текста. Полученное в результате столь утомительной процедуры 64-битовое число является функцией секретного ключа, начального вектора и каждого бита сообщения, вне зависимости от длины последнего. Это число получило условное обозначение MAC — message authentication code — и дописывается к зашифрованному сообщению, формируя таким образом как бы печать на конверте. Полученное «запечатанное» сообщение передается по каналу связи. Получатель расшифровывает сообщение и, зная секретный ключ и начальное значение вектора, вычисляет свое значение MAC. Если оба кода совпадают, то сообщение передано верно.

Но искушенный читатель, а как показывает редакционная почта, других у «Монитора» просто нет, наверняка уже заметил уязвимое место описанных вариантов действий. В самом деле, Алекс может сам убедиться, что присланное сообщение действительно отправлено Юстасом, но как ему доказать другим, например, радистке Кэт, что именно Юстас его отправил? Ведь поскольку и отправитель, и получатель сообщения пользуются при формировании аутентификатора одним и тем же ключом, получатель может подделать (да, читатель, бывают и такие коварные получатели!) и сообщение, и подпись отправителя.

В результате схема аутентификации пригодна для использования только между двумя группами взаимно доверяющих друг другу лиц, но не может быть использования для разрешения возникающих между ними противоречий. В частности, возникают неизбежные неприятности при постановке вопроса о передаче по линии связи платежных документов или информации, требующей нотариального заверения. Поэтому для усиления схемы аутентификации используют уже знакомые читателям однонаправленные функции, которые позволяют сформировать электронную подпись. Электронная подпись подразумевает, что Алекс сможет не только удостовериться сам, что сообщение отправлено Юстасом, но и, паче чаяния, доказать факт отправки сообщения именно Юстасом в суде!

Вообще-то говоря, цифровая подпись может быть реализована и в традиционных криптосистемах при использовании третьего участника, которому доверяют обе переговаривающиеся стороны. Этот третий участник, своеобразный «цифровой нотариус», был предложен Меркле (Merkle). Схема выглядит следующим образом.

Каждый пользователь системы (все тот же Алекс) регистрирует пару своих секретных преобразований E_a и D_a у «нотариуса» , причем, как и ранее, для любого сообщения m должно соблюдаться соотношение

$$E_a(D_a(m)) = m$$

Для того чтобы переслать подписанное сообщение m Юстасу, Алекс вычисляет $C = D_a(m)$ и передает его Юстасу. Чтобы проверить корректность с и получить открытый текст m, Юстас передает сообщение C «нотариусу», который вычисляет $E_a(C) = m$ и передает Юстасу m, зашифрованное личным ключом Юстаса $d = D_b(m)$.

В системах с открытым ключом электронная подпись реализована несколько элегантнее. Идея самой подписи относится еще к 1976 году, когда небезызвестные Вам Диффи и Хелманн предложили свою систему распределения ключей. В результате реализация электронной подписи возможна для любой системы с открытым ключом, для которой соотношение

$$E_a(D_a(m)) = m$$

выполняется для любого сообщения m.

Рассмотрим протокол формирования электронной подписи в такой системе. Предположим, Алекс пытается передать сообщение Юстасу.

1. Алекс вычисляет сигнатуру сообщения m, используя свой (!) личный ключ.

$$S = D_a(m)$$

2. Воспользовавшись затем общим ключом Юстаса, Алекс вычисляет

$$C = E_b(S)$$

и передает его Алексу.

3. На приемной стороне Юстас расшифровывает сообщение с помощью своего личного ключа D_b и получает сигнатуру S:

$$S = D_b(C)$$

4. Теперь, используя общий ключ Алекса, Юстас может восстановить исходный текст m:

$$m = E_a(S)$$

Теперь при возникновении спорной ситуации, в которой Алекс отказывается признать отправленное им сообщение, Алекс обязан предъявить арбитру свой личный ключ Da, удовлетворяющий требованию

$$D_a(E_a(X)) = X$$

для всех допустимых значений X.

Арбитру необходимо лишь проверить соотношение $D_a(m) = S$, и в случае его выполнения принимается решение, что никто, кроме Алекса, отправить сообщение не мог.

Кроме того, такой протокол обмена заставляет быть порядочным и Юстаса! Предположим, что Юстас исказил принятое сообщение m на m'. Тогда для того чтобы правда не вылезла наружу, необходимо изменить и сигнатуру S на S'. Но для этого нужен личный ключ Алекса, которого у Юстаса нет.

Подводя итоги, отметим, что реализация алгоритмов шифрования и электронной подписи, по мнению автора, представляет собой отдельную задачу, а поэтому редакция готова публиковать соответствующие материалы. Мы ждем Ваших писем! ■

Литература

1. В. Водолазкий. Коммерческие системы шифрования: основные алгоритмы и их реализация. Часть 2. Системы с открытым ключом// Монитор.– 1992.– № 8.
2. Trung Van Tran. Encipherment systems and autentication/ Доклад на конференции Kryptographie und Datensicherheit, 16.4-18.4.1991, Эссен, ФРГ.
3. Математика: Учебник для 5-го класса.– М: Просвещение, 1991.
4. Операция «Шнапс» и другие приключения Штирлица.– М., 1993.
5. В. Водолазкий. Коммерческие системы шифрования: основные алгоритмы и их реализация. Часть 1// Монитор.– 1992.– № 6-7.

Русскоязычная версия!

Just Write™

SYMANTEC

Диалог-МИФИ
095-320-3466

Merisel-Russia
095-276-9008

Параграф
095-299-7923

Перспективные технологии
095-256-6271

Трио Плюс
095-971-1204

Прекрасный текстовый процессор...
Для Windows™...
Для тех, кто хочет быстро, просто и красиво работать с текстом...

Книги и журналы, микрофиши и фотографии, чертежи любого формата, станки, машины, здания, самолеты, пароходы, красивые девушки, фрукты, пляжи, отпуск на Багамск... ...вах, дорогие рестораны, оте... ...сти, драгоценности, ша... ...ном, негр-официа... и бель... кра... ...ребро сир... ...рской пены, цвет... ...еревитые ленточкой и хру... ...венецианского стекла

Все сможет отсканировать проекционный сканнер ScanNex

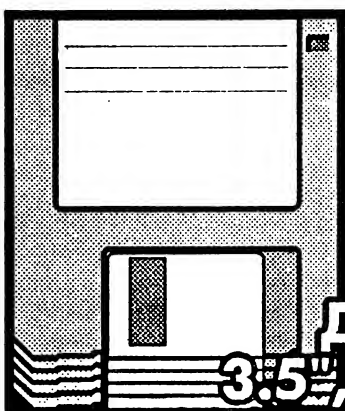
Компания

ExTel

256 оттенков серого,
разрешение: 300 точек/дюйм для формата А4,
2900 точек/дюйм для микрофильмов.
Гарантия — два года

(095) 117-8215

(095) 258-3776
(095) 111-3068
(8312) 36-4650



ОТЛИЧНЫЕ ЦЕНЫ! УБЕДИТЕСЬ САМИ

ДИСКЕТЫ 3.5", 1.44Mb, 2S/HD TOYSAN TECHNOLOGIES (США)

телефоны:
в Минске (0172) 295821
в Москве (095) 2550523

Крупнейшая информационная сеть СНГ даст Вам возможность общения с любым абонентом всемирных сетей Internet, Bitnet, MCI-Mail, Compuserv, EUnet и многих других.

Ваше сообщение, программа или графический файл дойдет до любой точки мира меньше чем за 4часа! В отличие от факса, не будет потеряно ни одной буквы.

Электронная почта— современнейшая технология общения. Теперь она доступна каждому.

Используя возможности сети RELCOM, Вы можете обсудить проблему со всеми заинтересованными в ее решении людьми по всей планете одновременно, Вы можете извлечь информацию из сотен банков данных, получить свежую версию программы, даже рекламировать свою продукцию и услуги.

В сети RELCOM активно работают многие биржи, банки, крупные и небольшие предприятия, институты— всего более 60,000 на территории СНГ и далеко за миллион во всем мире. У абонентов RELCOM появляется возможность прямой связи с большинством крупных компьютерных, программистских и электронных фирм.

Единственный недостаток RELCOM — однажды начав работать в ней, Вы будете делать это всю жизнь.

Телефоны: (095) 231-21-29, 231-63-95, 233-05-92, 233-02-42, 231-60-02
Факс: (095) 233-50-16
E-mail: info@hq.demos.su

Relcom является зарегистрированной торговой маркой фирмы Демос

Эффекты для демонстрации слайдов

Е. Кисель

Побудительным мотивом к написанию программного модуля, рассматриваемого в этой статье, для автора послужили два события: заказ на разработку небольшого рекламного ролика и публикация в журнале «Монитор» статьи А.И.Тихонова «Показать картинку...».

Заказанный рекламный ролик не требовал привлечения сложных механизмов типа мультипликации и мог быть реализован в виде простой последовательности слайдов, выводимых на экран. В дальнейшем такую последовательность мы будем называть слайд-фильмом. При этом под слайдом подразумевается графический образ (картинка), имеющий прямоугольную форму и не превышающий по размерам экрана дисплея. Для оживления подачи материала и привлечения внимания к отдельным моментам изложения слайды должны появляться на экране с использованием различных изобразительных приемов: раздвигание границ, наплыв и так далее. Эти приемы мы будем называть эффектами.

В общем виде постановку задачи можно сформулировать так: мы хотим научиться писать программы, отображающие на экране последовательности слайдов с использованием эффектов.

Оговоримся, что предлагаемая статья никоим образом не задумывалась как дидактическое пособие по написанию слайд-фильмов. Главное, что автору хотелось донести до читателя, — это то, что многие внешне сложные продукты вполне по силам программисту не самой высокой квалификации. Поэтому, прежде чем приобретать ту или иную универсальную систему, стоит задуматься, а не дешевле ли для Вашей конкретной задачи разработать собственный инструментарий. За универсальность зачастую приходится платить очень дорого и не только деньгами. Короче говоря, при начале нового проекта Вам каждый раз необходимо решить далеко не тривиальный треугольник: универсальность/эффективность/затраты.

Исходя из приведенных соображений, попробуем выбрать используемые инструментальные средства. Самым простым решением вопроса, наверное, является выбор одной из таких известных систем, как StoryBoard, ShowPartner или даже CorelDraw. Однако бедному (в прямом смысле слова) отечественному программисту они не по карману, а использование нелицензионных копий программ в коммерческих целях в нашей стране наконец-то преследуется по закону. С другой стороны, перечисленные продукты слишком громоздки для небольших приложений и полученные слайд-фильмы не могут быть встроены в программу на языке C или Pascal, что во многих случаях было бы желательно. Альтернативный (и более предпочтительный) вариант — выбор библиотеки PCX Effects фирмы Genus.

Библиотека PCX Effects предоставляет программисту инструментарий для формирования слайд-фильмов без необходимости использования резидентных или интерпретирующих модулей. Библиотека реализована для компиляторов с языка C фирм Microsoft, Borland и Lattice, с языка Pascal фирм Microsoft и Borland, с языков Assembler, Fortran и Basic фирмы Microsoft и даже для компилятора Clipper фирмы Computer Associates. Перечень эффектов, обеспечиваемых библиотекой PCX Effects, с их краткой характеристикой представлен ниже в глоссарии. За смысловую нагрузку названий эффектов автор ответственности не несет. Поэтому, воспользовавшись словарем проф. В.К.Мюллера, он снабдил соответствующие имена короткими справка-

ми, включающими их наиболее употребимые значения. О соответствии имен тому, что ими названо, судите сами.

К сожалению, данная библиотека требует использования библиотеки PCX Programmer's Toolkit, не поддерживающей Turbo Pascal версий старше 5.0. В описанной ситуации нам ничего не остается, кроме как пойти по пути, проторенному А.И. Тихоновым, — попытаться воспроизвести следующий за PCX Programmer's Toolkit продукт — библиотеку PCX Effects. При этом процедуры чтения файлов в формате PCX можно взять, естественно, с разрешения автора, из статьи «Показать картинку...».

Воспроизведение (remake) в программировании — такое же часто встречающееся явление, как и в искусстве кино. Правда, в отличие от кинопродюсера, программист обычно лишен возможности ознакомиться со «сценарием» прототипа, и вынужден руководствоваться лишь «внешним видом» воссоздаваемого произведения.

Опишем внешние спецификации разрабатываемого модуля. Если бы мы начинали на пустом месте, этот этап мог бы растянуться на месяц-другой, но мы ведь воспроизводим известный продукт! Поэтому спасибо фирме Genus, проделавшей за нас «нулевой цикл» разработки.

Во-первых, модуль должен отображать на экране графические образы с использованием специальных эффектов.

Во-вторых, он должен быть достаточно универсальным, чтобы его можно было использовать на широком классе графических адаптеров и конкретных конфигураций ПЭВМ. Это требование легко удовлетворить, если использовать графические примитивы, предоставляемые драйверами Borland Graphics Interface (BGI), входящими в стандартный комплект поставки компиляторов фирмы Borland.

На универсальности следует остановиться особо, поскольку в предлагаемом программном модуле использована маленькая хитрость (кое-кто, возможно, назовет это ленью), которая исключает возможность работы с графическим адаптером CGA или EGA с объемом видеопамяти менее 256 Кб. Дело в том, что графический образ перед показом предварительно формируется на невидимой странице экрана, что подразумевает наличие этой самой страницы. Но я считаю, что это ограничение не так уж существенно, поскольку демонстрация слайд-фильмов на CGA с эстетической точки зрения не удовлетворит даже самого невзыскательного зрителя, а адаптеры EGA со 128 Кб и тем более с 64 Кб видеопамяти в основном отошли в прошлое.

В-третьих, наш модуль должен предоставлять возможность управления скоростью демонстрации фильма.

В-четвертых, так как размер показываемого слайда может быть меньше, чем экран дисплея, необходимо предусмотреть возможность изменения его положения относительно границ экранной страницы.

Глоссарий

Wipe — 1. *v* вытирать, утирать; to w. the floor with — изничтожить кого-либо; 2. *n* вытирание; удар с размаху.
Эффект Wipe заключается в стирании с экрана предыдущего изображения и замещении новым в направлении сверху вниз.

Split — 1. *v* раскалывать(ся); расщеплять(ся); 2. *n* раскалывание; трещина; раскол.
Эффект Split представляет собой «раздвигание» границ слайда по вертикали от центра.

Crush — 1. *v* раздавливать; выжимать; втискивать; 2. *n* раздавливание; давка.
Эффект Crush является своего рода обратным действием по отношению к эффекту Split. Внешне это выглядит как «сдвигание» шторок с выводимым изображением.

Strips — ленты; полоски.
При эффекте Strips на экран с некоторыми промежутками последовательно выводятся горизонтальные полоски слайда. Подобным образом часто кодируются графические файлы в формате GIF, так что внешнее проявление этого эффекта Вам, скорее всего, знакомо.

Slide — 1. *v* скользить; 2. *n* скольжение; оползень.
Эффект Slide — один из любимых эффектов автора. Внешне это выглядит так, как будто картинка постепенно выползает из щели на экране.

Drip — 1. *v* капать, падать каплями; 2. *n* капанье.

Sand — 1. *n* песок; гравий; 2. *v* посыпать песком.
Эффекты Drip и Sand также очень красивы, они, как и эффекты Split и Crush, взаимно дополняют друг друга и отличаются лишь направлением вывода (сверху вниз и снизу вверх соответственно). Может быть, у кого-то они и вызывают ассоциации с каплями и песком, автору же они скорее напоминают работу ткацкого станка, особенно если рисунок имеет периодическую структуру, например, представляет собой какой-либо узор или решетку.

Explode — *v* взрывать(ся); разбивать, подрывать.
Эффект Explode можно рассматривать как одновременное выполнение эффекта Split по горизонтали и вертикали, при этом слайд начинает раздвигаться от центра в виде креста.

Diagonal — 1. *a* диагональный, идущий наискось; 2. *n* диагональ.
При эффекте Diagonal слайд выводится по диагонали, начиная от верхнего левого и заканчивая нижним правым углом.

Spiral — 1. *a* спиральный; винтовой; 2. *n* спираль.
Эффект Spiral заключается в выводе слайда по спирали от центра.

Random — 1. *n* at r. наугад, наобум, наудачу; 2. *a* сделанный или выбранный наугад, случайно.
При эффекте Random отдельные участки слайда выводятся на экран в случайном порядке.

В-пятых, необходимо предусмотреть «дробность» проявления эффекта, то есть знать, какого размера прямоугольники и какой толщины линии будут использоваться.

Все перечисленные требования, кроме выравнивания положения фрагмента относительно границ экрана и, естественно, использования BGI, в той или иной мере учтены в библиотеке PCX Effects фирмы Genus.

Определив мир сущностей, которыми мы собираемся манипулировать в разрабатываемом модуле, попробуем описать теперь операции, которые нам хотелось бы уметь осуществлять:

1. Мы бы хотели отображать графический фрагмент на экране с заданным эффектом и в указанной позиции.
2. Иногда, устав от мелькания, мы были бы рады вывести на экран картинку без эффектов.
3. Коль скоро мы используем константы выравнивания, не мешало бы ввести процедуру для их установки.
4. Иногда бывает полезно проверить, возможно ли выполнение наших «фокусов» на наличных технических средствах, для этого введем процедуру инициализации эффектов (и соответствующую ей логическую константу).

Теперь мы готовы написать интерфейсную секцию модуля в строгом соответствии с технологией программирования сверху вниз. (Конечно автор писал модуль методом «последовательных трансформаций», замечательно описанным Стругацкими в сказке для научных сотрудников младшего возраста, но почему бы не порисоваться.) Что у нас получилось в итоге, Вы можете обнаружить в листинге 1.

Не хочу быть навязчивым, но если Вам необходим маленький комментарий, то вот и он! Первая загадочная строка модуля устанавливает для него своеобразное распределение памяти. Ее необходимость обусловлена использованием в процедуре для эффекта Random, временного массива большого объема. Перечислимый тип Effects задает символьные обозначения для эффектов демонстрации, а тип Justifications — для горизонтального и вертикального выравнивания. Все наименования эффектов позаимствованы из библиотеки PCX Effects, исключение составляет лишь эффект Strips — полоски. До реализации этого эффекта у программистов фирмы Genus, видимо, просто руки не дошли. Константы ScrWidth и ScrHight введены для того, чтобы поменьше обращаться к соответствующим функциям BGI. Использование модуля Objects объясняется пристрастным отношением автора к библиотеке TurboVision и желанием использовать тип TPoint для задания координат экрана. Почему функция ShowWithEffect возвращает указатель (в описываемой версии модуля почти всегда NIL), мы поймем из заключения, где рассмотрим возможности дальнейшего развития.

Если длительностью присутствия слайда на экране можно управлять с помощью процедуры Delay (модуль CRT), то задержку для времени выполнения эффекта появления графического образа на экране нужно учитывать отдельно — для этого введена константа CurrentDelay.

Очевидный путь решения для изменения положения выводимого слайда — задание координат верхнего левого угла отображаемого фрагмента. Именно так поступили разработчики библиотеки PCX Effects, но даже при воспроизведении системы в программировании всегда найдется место для творчества. Если немного поразмыслить, то окажется, что в большинстве случаев нам хотелось бы располагать отображаемый фрагмент либо по центру экрана, либо по его углам, либо в некоторых промежуточных положениях относительно указанных позиций. Для этого достаточно ввести константы выравнивания по горизонтали и вертикали CurrentJust, указывающие, что слайд должен занимать положение по вертикали верхнее (Top), нижнее (Bottom) или центральное (Center) и по горизонтали левое (Left), правое (Right) или центральное (Center).

Несколько вспомогательных процедур, приведенных в листинге 2, значительно облегчат нам работу над основным телом модуля.

Использование модуля Graph необходимо для доступа к графическим примитивам BGI. Модуль PcxPic без изменений взят из статьи «Показать картинку...» и обеспечивает чтение файлов в формате PCX. Из модуля CRT мы используем процедуру Delay для задания временных задержек.

Применение процедуры EffectsInit гарантирует, что даже при отсутствии невидимой графической страницы Вы все-таки увидите слайд на экране, хотя бы и без использования эффекта.

Процедура SetJustify настолько проста, что в комментариях не нуждается. Несколько сложнее последствия установки констант выравнивания. Для обеспечения их обработки приходится вводить вспомогательные процедуры GetDimentions и Justify.

Простейший эффект — никаких эффектов! Для его реализации мы используем процедуру DirectShow, которая просто вызывает процедуру ShowPcxFromFile из модуля PcxPic. При этом механизм выравнивания позиции выводимого слайда включается при задании нулевых координат верхнего левого угла. Это никоим образом не помешает Вам при необходимости выводить фрагменты изображения в любой другой точке экрана. Если же Вы хотите вывести слайд в позиции (0,0), необходимо просто сделать установку выравнивания SetJustify(Left,Top).

Еще две простенькие процедуры очень пригодятся нам при реализации эффектов для демонстрации слайдов: CopyBlock и CopyGrain, представленные в листинге 3.

Рассмотрим теперь функцию, ради которой автор так долго испытывает Ваше терпение — ShowWithEffect. Ее упрощенная схема представлена в листинге 4. Переключение активной страницы экрана указывает, что обмен по вводу-выводу в дальней-

шем будет осуществляется с использованием участка видеопамяти, соответствующего видимой или невидимой странице. В нашем случае для простоты подразумевается, что видимая страница — нулевая, а невидимая — первая.

Вот и все! (Как любят говорить на телевидении.) Но в самом деле тут-то и начинается самое интересное. Посмотрим еще раз расстановку действующих лиц перед решающим сражением. Мы имеем:

- на невидимой (первой) странице в позиции (0,0) у нас подготовлен слайд, который мы хотим показать;
- позиция экрана, в которую мы хотим перенести слайд на видимую (нулевую) страницу, описывается переменной Loc — (Loc.X,Loc.Y);
- размеры слайда — Dim.X на Dim.Y пикселов;
- дробность проявления эффекта — Grain пикселов;
- размеры экрана — ScrWidth на ScrHight пикселов.

Итак, применим эффекты!

Эффект Wipe (листинг 5) — это уже надоевший нам прямой вывод на экран. Процесс вывода заключается в чтении файла, построчной распаковке и выводе полученных строчек на экран сверху вниз.

Для эффекта Split (листинг 6) мы последовательно копируем по две полосы шириной Grain пикселов, одна из которых выше, а другая ниже уже отображенного участка. Цикл начинается от центральной разделительной горизонтальной линии и заканчивается по достижении границ слайда.

При эффекте Crush (листинг 7) копирование полос начинается от верхней и нижней границ слайда и заканчивается по достижении центральной разделительной горизонтальной линии. Текст данного участка программы лишь незначительно отличается от предыдущего.

Эффект Strips (листинг 8) — еще один «полосатый» эффект. Правда, полоски шириной в один пиксел — это скорее строчки или нити, но пытливый читатель без труда изменит код таким образом, чтобы дробность эффекта учитывалась и в этом случае. Алгоритм эффекта Strips заключается в следующем: полоски слайда, начиная с верхней, с шагом в пять полосок, копируются на экран, после достижения нижней границы слайда переходим ко второй сверху полоске и повторяем процесс. Цикл с постоянным смещением верхней копируемой полоски повторяем пять раз (от нулевой до четвертой полоски).

При всей изобразительной привлекательности эффекта Slide (листинг 9) его реализация чрезвычайно проста: мы копируем все более и более широкие полосы слайда, начиная с его нижней границы, в заданную позицию экрана. К сожалению, при этом объем памяти, необходимой для хранения слайда, не должен превышать 64 Кбайтов, что обусловлено использованием BGI.

Для достижения эффектов Drip и Sand (листинг 10) мы опять используем полоски шириной Grain пикселов. Берем первую полоску (сверху для эффекта Drip или снизу для Sand) и заполняем ею всю область экрана, предназначенную для слайда. Первая полоска теперь на своем месте. Переходим ко второй полоске и заполняем область слайда, уменьшенную на ширину полоски, соответственно сверху или снизу. Повторяем так до тех пор, пока весь слайд не будет скопирован на экран. То, что потребовало так много слов для описания, достаточно кратко записывается на языке Turbo Pascal (см. листинг).

При реализации эффекта Explode мы сталкиваемся с новым парадоксом — то, что так просто выразить словами, требует несколько больше строк на языке программирования (листинг 11).

От «полосатых» перейдем к «ячеистым» эффектам. Первый из них — Diagonal представлен в листинге 12. Обратите внимание, что теперь вместо процедуры CopyBlock мы будем использовать процедуру CopyGrain. В этом и последующих эффектах мы будем рассматривать слайд как двумерный массив прямоугольников (ячеек) Grain на Grain пикселов. Такой взгляд на вещи значительно упрощает решение задачи. Нам необходимо только задать порядок, в котором копировать ячейки. Для эффекта Diagonal на каждой итерации цикла копируются ячейки, занимающие, как следует из названия, диагональное положение, начиная от верхней левой и заканчивая нижней правой.

После удачно выбранной «ячеистой» парадигмы реализация эффекта Spiral (листинг 13) не требует особых интеллектуальных усилий. Ячейки копируются по спирали, начиная с центральной.

Эффект Random — один из наиболее сложных в реализации, как и все, что связано со случайностью. Задача состоит в том, чтобы скопировать слайд с невидимой страницы на экран, выбирая ячейки случайным образом. Первая проблема заключается в самом слове «случайный». В самом деле, истинно случайную последовательность чисел средствами цифровой вычислительной техники создать практически невозможно. Поэтому приходится пользоваться различного рода суррогатами, например, псевдослучайной последовательностью, которую возвращает функция Random из модуля System. Вторая проблема в том, что псевдослучайная последовательность, возвращаемая функцией Random, моделирует выборку с возвратом. Это значит, что однажды появившийся в последовательности элемент может повторяться в ней любое количество раз, тогда как некоторый другой может не появиться никогда. Ясно, что, руководствуясь такими принципами, мы не можем гарантировать, что процесс копирования ячеек когда-либо закончится. Автор не знает, как реализован эффект Random в библиотеке PCX Effects, но опытная проверка показала, что описываемый ниже алгоритм работает значительно быстрее.

Идея заключается в том, чтобы не разыгрывать появление каждого следующего элемента по мере копирования (при выборке без возврата, то есть когда каждый элемент в последовательности может появиться только один раз, это потребует гораздо большего количества проверок на каждом шаге), а генерировать

всю последовательность сразу. Для этого нам потребуется еще одна вспомогательная процедура PrepearRand (листинг 14).

Магическое число 8959 представляет из себя количество ячеек 5 на 5 пикселов при разбивке полного экрана в режиме 640 на 350 и выбрано автором в качестве компромиссного решения для эффективной обработки достаточно больших образов. Собственно ради передачи сформированного массива в качестве параметра рассмотренной процедурой нам и понадобилась первая строка модуля, задающая распределение памяти.

Теперь реализация эффекта Random почти тривиальна (листинг 15).

Пример использования сконструированного нами модуля SlideEffects представлен в листинге 16.

Заключение, или к чему все это и что же дальше?

Вернемся еще раз к представленному программному модулю. Какие возможности для его развития может обнаружить пытливый ум читателя?

Прежде всего, мы воссоздали библиотеку PCX Effects не в полном объеме. Например, мы упустили такую характеристику эффекта, как его направление. Это сделано автором не из-за неимоверной сложности реализации, а исключительно в целях экономии бумаги.

После этого можно устранить ограничения, связанные с требованием наличия невидимой графической страницы, что существенно расширит класс приемлемых технических средств. Для этого необходимо решить следующие сопутствующие вопросы:

- разработка процедур размещения в оперативной памяти графических образов и перенос их на экран, причем стандартные процедуры GetImage и PutImage нас не устроят, так как они не обеспечивают прямое обращение к соответствующим участкам образа по относительным координатам;
- разработка процедур для манипуляций с участками динамической памяти, превышающими 64 Кбайта (это, кстати, еще одно причина, почему GetImage и PutImage неприменимы).

Как первые, так и вторые реализуются достаточно очевидно при минимальных знаниях о технических средствах и механизмах управления динамической памятью в Turbo Pascal.

Очень просто расширить наш модуль для использования графических форматов, отличных от PCX. Необходимо только уметь читать соответствующий формат и определять размеры записанного в файле фрагмента.

Далее, очень часто небольшие фрагменты изображения в слайд-фильмах последовательно появляются и исчезают на одном и том же фоне. В предлагаемой реализации у нас нет другого выхода, кроме как восстанавливать каждый раз полный экран из файла. Более эффективный способ заключается в сохранении участка экрана, закрываемого новым фрагментом, в динамической памяти и последующем восстановлении этого участка без обращения к диску. Именно для этого предусмотрено возвращение указателя функцией ShowWithEffect. В более общем случае список перекрываемых фрагментов можно организовать в виде стека, для реализации которого идеально подходит объект TCollection из библиотеки TurboVision, тем более, что мы к ней уже обращались для использования типа TPoint.

Описанные изменения дадут нам в результате практически полный набор базовых процедур для реализации графического интерфейса с пользователем, но это уже другая история, а поскольку, как говорили древние, «Vita brevis, ars longa», необходимо уметь вовремя остановиться и поставить точку. ■

ЛИТЕРАТУРА

1. Тихонов А.И. Показать картинку...// Монитор.– 1991.– № 3.
2. Стругацкий А., Стругацкий Б. Понедельник начинается в субботу. — Фрунзе: Мектеп, 1987.
3. Мюллер В.К. Англо-русский словарь.– М.: Государственное издательство иностранных и национальных словарей, 1950.
4. Романов В.Ю. Популярные форматы файлов для хранения графических изображений на IBM PC.– М.: Унитех, 1992.
5. Wilton R. The programmer's guide to PC and PS/2 video systems.– Redmond: Microsoft Press, 1987.

Монитор — 1994

Подписной купон

МОНИТОР

1	2	3	4	5	6	7	8

Адрес _____

Ф.И.О. или название организации _____

Стоимость подписки на 1 номер журнала "Монитор" для России — 600 рублей, для стран СНГ — 900 рублей. В 1994 году будет выпущено 8 номеров (годовая подписка — соответственно 4800 руб. и 7200 руб.). Отметьте крестиками те номера журнала, которые Вы хотите получить. Перечислите сумму оплаты (действительна до 31 декабря 1993 г.) на счет НТО "Софт-Москва": р/с 467002 в КБ "Банк развития XXI век", к/с 161816 в РКЦ ГУ ЦБ РФ по г. Москве, МФО 201791.

Вышлите подписной купон и квитанцию об оплате по адресу: 117419 Москва, а/я 765.

Листинг 1. Интерфейсная секция модуля для демонстрации слайдов с использованием эффектов

```
{$M 65520,0,655360}

{*******************************************************
    unit SlideEffects — процедуры демонстрации слайдов на основе
                        графических примитивов BGI.
                    Copyright (c) 1993, Е.Кисель
*******************************************************}

unit SlideEffects;

interface

uses    Objects;

type
 Effects = (Wipe, Split, Crush,          { Типы эффектов.}
            Slide, Sand, Drip,
            Explode, Spiral, Diagonal,
            Random, Strips);

 Justifications = (Center, Top, Bottom,  { Типы выравнивания.}
                   Left, Right);

const
 Grain: integer = 5;                     { Дробность эффекта.}
 CurrentEffect: Effects = Wipe;          { Текущий эффект.}
 CurrentDelay: word = 1;                 { Текущая задержка.}
 CurrentJust: array [1..2] of            { Текущее выравни-}
    Justifications =                     { вание. По умолча-}
    (Center,Center);                     { нию — центр.}
 ScrWidth = 640;                         { Ширина экрана.}
 ScrHight = 350;                         { Высота экрана.}
 UseEffects: boolean = TRUE;             { Применять эффект?}

{EffectsInit — инициализация эффектов.}
procedure EffectsInit;

{ShowWithEffect —  показать слайд из файла ImageFl в позиции
                   экрана Location с использованием текущего
                   эффекта.}
function ShowWithEffect(ImageFl: string; Location: TPoint): pointer;

{DirectShow —      показать слайд из файла ImageFl в позиции экрана
                   Location напрямую.}
procedure DirectShow(ImageFl: string; Location: TPoint);

{SetJustify — установить режимы выравнивания.}
procedure SetJustify(XJust,YJust: Justifications);
```

Листинг 2. Вспомогательные процедуры модуля SlideEffects

```
implementation

uses Graph, PcxPic, Crt;

{EffectsInit — инициализация эффектов.}
procedure EffectsInit;
var
 GraphDriver, GraphMode: Integer;
begin
 DetectGraph(GraphDriver, GraphMode);          { Проверка текущего }
 case GraphDriver of                           { графического режима }
    EGA: UseEffects:= True;                    { и присваивание }
    VGA: if  (GraphMode = VGALo) or
             (GraphMode = VGAMed)              { корректного }
       then UseEffects:= True;                 { значения константе }
    HERCMono: UseEffects:= True;               { UseEffects }
 else UseEffects:= False;
 end;
end;

{SetJustify — установить режимы выравнивания.}
procedure SetJustify(XJust,YJust: Justifications);
begin
 CurrentJust[1]:= XJust;
 CurrentJust[2]:= YJust;
end;

type
 PcxHeader = Record       { Заголовок PCX файла (усеченный), }
                          { нужен для определения размерности слайда }
 Creator: Byte;           { ZSoft = 10 }
 Version: Byte;           { версия PCX }
 Encoding: Byte;          { 1 — тип сжатия RLE }
 Bits: Byte;              { число битов на пиксел }
 Xmin,                    {********************}
 Ymin,                    {** координаты углов }
 Xmax,                    {** слайда }
 Ymax:integer;            {********************}
end;

procedure GetDimentions(ImageFl: string;var Dim: TPoint);
var
 PCXFl: File;
 Header: PCXHeader;
 s,size: word;
 i: integer;
begin
 {$I-}
 Assign(PCXFl,ImageFl);
 i:= Ioresult;
 if i <> 0 then
 begin
    Dim.X:= 0;
    Dim.Y:= 0;
    Exit;
 end;
 {$I+}
 Reset(PCXFl,1);
 Size:= SizeOf(Header);                  { Чтение усеченного заголовка }
 BlockRead(PCXFl,Header,Size,s);         { PCX файла для определения }
 Dim.Y:= Header.Ymax-Header.Ymin;        { размеров выводимого слайда }
 Dim.X:= Header.Xmax-Header.Xmin;        { Dim.X и Dim.Y}
 Close(PCXFl);
end;

procedure Justify(var Dim: TPoint);
var
 TempPoint: TPoint;
begin
 if Dim.X >= ScrWidth then TempPoint.X:= 0          { Процедура Justify, }
 else case CurrentJust[1] of                        { исходя из размеров }
    Center: TempPoint.X:= (ScrWidth-Dim.X) div 2;   { слайда Dim, текущих }
    Left: TempPoint.X:= 0;                          { установок выравнивания }
    Right: TempPoint.X:= ScrWidth-Dim.X;            { CurrentJust и размеров }
 end;                                               { экрана ScrWidth и }
 if Dim.Y >= ScrHight then TempPoint.Y:= 0          { ScrWidth вычисляет }
 else case CurrentJust[2] of                        { координаты желаемого }
    Center: TempPoint.Y:= (ScrHight-Dim.Y) div 2;   { положения слайда на }
    Top: TempPoint.Y:= 0;                           { экране. }
    Bottom: TempPoint.Y:= ScrHight-Dim.Y;
 end;
 Dim:= TempPoint;
end;

{DirectShow — показать слайд из файла ImageFl в позиции экрана
              Location напрямую.}
procedure DirectShow(ImageFl: string; Location: TPoint);
var
 Dim : TPoint;
begin
 if (Location.X = 0) and (Location.Y = 0) then
 begin
    GetDimentions(ImageFl,Dim);
                                  { Определяем размеры выводимого слайда }
    Justify(Dim);                 { Применяем процедуру выравнивания }
    Location:= Dim;               { позиции вывода }
 end;
 ShowPcxFromFile(Location.X,Location.Y,3,True,NormalPut,ImageFl)
end;
```

Листинг 3. Процедуры межстраничного копирования отдельных участков выводимого изображения

```
{ CopyBlock копирует прямоугольник размером DimX на DimY
  из позиции (TopX,TopY) невидимой графической страницы
  в позицию экрана (NewX,NewY) }
procedure CopyBlock(TopX,TopY,NewX,NewY,DimX,DimY: integer);
var
 MemBuff: Pointer;
 ImgSz: word;
begin
 ImgSz:= ImageSize(0,0,DimX,DimY);
 If ImgSz = 0 then Exit;
 GetMem(MemBuff,ImgSz);
 SetActivePage(1);
 GetImage(TopX,TopY,TopX + DimX,TopY + DimY,MemBuff^);
 SetActivePage(0);
 PutImage(NewX,NewY,MemBuff^,NormalPut);
 FreeMem(MemBuff,ImgSz);
end;
```

```
{ CopyGrain копирует ячейку Grain на Grain пикселов.
    Pnt задает координаты
    копируемого фрагмента не в пикселах, а в ячейках. Loc указывает
    положение верхнего левого угла всего слайда. Wd и Hd — ширина и
    высота слайда соответственно }
procedure CopyGrain(Pnt,Loc: TPoint; Wd,Hd: integer);
var
 TopPt,BotPt: TPoint;
begin
 TopPt.X: = Pnt.X*Grain;
 If word(TopPt.X) > word(Wd) then exit; { Указание размеров слайда }
 TopPt.Y: = Pnt.Y*Grain;                { необходимо для правильной }
 If word(TopPt.Y) > word(Hd) then exit; { обработки границ слайда в тех }
 BotPt.X: = (Pnt.X + 1)*Grain;          { случаях, когда он не делится на }
 If BotPt.X > Wd then BotPt.X: = BotPt.X-Wd { целое число ячеек.}
 else BotPt.X: = Grain;
 BotPt.Y: = (Pnt.Y + 1)*Grain;
 If BotPt.Y > Hd then BotPt.Y: = BotPt.Y-Hd
 else BotPt.Y: = Grain;
 CopyBlock(TopPt.X,TopPt.Y,Loc.X + TopPt.X,Loc.Y + TopPt.Y,BotPt.X,BotPt.Y
);
end;
```

Листинг 4. Упрощенная схема функции ShowWithEffect

```
{ShowWithEffect — показать слайд из файла ImageFl в позиции
                  экрана Location с использованием текущего
                  эффекта.}
function ShowWithEffect(ImageFl: string; Location: TPoint): pointer;
var
 Error: integer;
 I, J, J1, I1, K: word;
 Loc, Cntr, Dim: TPoint;
 RandomArea: array [0..8959] of word;
begin
 if not UseEffects then        { Проверяем, можно ли использовать эффекты, }
 begin                         { и если нельзя, то ... }
    DirectShow(ImageFl,Location); { показываем слайд напрямую }
    Exit;                      { и на этом заканчиваем. }
 end;
 GetDimentions(ImageFl,Dim);   { Определяем размеры слайда }
 if (Location.X = 0) and (Location.Y = 0) then
 begin
    Loc.X: = Dim.X;            { Выравниваем положение слайда }
    Loc.Y: = Dim.Y;            { относительно границ экрана }
    Justify(Loc);
 end
 else Loc: = Location;
 SetActivePage(1);             { Переключаем активную страницу экрана }
 If CurrentEffect < > Wipe then { Для любого эффекта, кроме Wipe, }
 begin                         { читаем графический образ из файла }
    Error: = ShowPcxFromFile(0,0,3,True, { ImageFl на невидимую страницу }
          NormalPut,ImageFl);
    if Error < > 0 then        { Если что-нибудь не так, заканчиваем выполнение }
    begin
       ShowWithEffect: = nil;
       SetActivePage(0);
       Exit;
    end;
 end;
 SetActivePage(0);    { Возвращаем активность видимой странице экрана }
 case CurrentEffect of  { Распознаем, какой эффект нам требуется, и ... }
    Wipe:     {...} ;
    Split:    {...} ;
    Crush:    {...} ;
    Strips:   {...} ;
    Slide:    {...} ;  { ... выполняем соответствующую CASE ветку. }
    Drip:     {...} ;
    Sand:     {...} ;
    Explode:  {...} ;
    Diagonal: {...} ;
    Spiral:   {...} ;
    Random:   {...} ;
 else ShowWithEffect: = nil;
 end;
end;
```

Листинг 5. Эффект Wipe

```
Wipe: begin     { Прямой вывод слайда на видимую страницу }
 ShowPcxFromFile(Loc.X,Loc.Y,3,True,NormalPut,ImageFl);
 Delay(CurrentDelay);
 ShowWithEffect: = nil;
end;
```

Листинг 6. Эффект Split

```
Split: begin
 J: = Dim.Y div 2;       { Определение центральной горизонтали слайда }
 I: = 0;
 repeat
    CopyBlock(0,J + I,Loc.X,Loc.Y + J + I,Dim.X,Grain); { Копирование верхней и }
    CopyBlock(0,J-I,Loc.X,Loc.Y + J-I,Dim.X,Grain);  { нижней полос }
    Delay(CurrentDelay);                 { Задержка в CurrentDelay, мс }
    Inc(I,Grain);
    if (I > J) and ((I-J) < Grain) then I: = J-Grain;
 until I > J;                            { Границы слайда достигнуты? }
end;
```

Листинг 7. Эффект Crush

```
Crush: begin
 J: = Dim.Y div 2;       { Определение центральной горизонтали слайда }
 I: = 0;
 repeat
    CopyBlock(0,I,Loc.X,Loc.Y + I,Dim.X,Grain);
    CopyBlock(0,Dim.Y-I,Loc.X,Loc.Y + Dim.Y-I,Dim.X,Grain);
    Delay(CurrentDelay);                 { Задержка в CurrentDelay, мс }
    Inc(I,Grain);
    if (I > J) and ((I-J) < Grain) then I: = J-Grain;
 until I > J;              { Центральная горизонталь слайда достигнута? }
end;
```

Листинг 8. Эффект Strips

```
Strips: begin
 for J: = 0 to 4 do
 begin
    I: = J;
    While I < = Dim.Y do
    begin
       CopyBlock(0,0 + I,          { Копирование каждой пятой строчки }
          Loc.X,Loc.Y + I,         { слайда, начиная с I-й }
          Dim.X,1);
       Inc(I,5);
    end;
    Delay(CurrentDelay);           { Задержка в CurrentDelay, мс }
 end;
end;
```

Листинг 9. Эффект Slide

```
Slide: begin
 for I: = 1 to (Dim.Y div Grain) do
 begin
 { Копирование полос, увеличивающихся по ширине }
    CopyBlock(0,Dim.Y-I*Grain,Loc.X,Loc.Y,Dim.X,I*Grain);
    Delay(CurrentDelay);                 { Задержка в CurrentDelay мс }
 end;
end;
```

Листинг 10. Эффекты Drip и Sand

```
Drip: begin
 for I: = 0 to ((Dim.Y-1) div Grain) do
 begin
    for J: = ((Dim.Y-1) div Grain) downto I do
 { Заполнение оставшейся части слайда текущей полоской }
       CopyBlock(0,I*Grain,Loc.X,Loc.Y + J*Grain,Dim.X,Grain);
    Delay(CurrentDelay);                 { Задержка в CurrentDelay, мс }
 end;
end;

Sand: begin
 for I: = ((Dim.Y-1) div Grain) downto 0 do
 begin
    for J: = 0 to I do
 { Заполнение оставшейся части слайда текущей полоской }
       CopyBlock(0,I*Grain,Loc.X,Loc.Y + J*Grain,Dim.X,Grain);
    Delay(CurrentDelay);                 { Задержка в CurrentDelay, мс }
 end;
end;
```

Листинг 11. Эффект Explode

```
Explode: begin
 Cntr.X: = Dim.X div 2;                  { Центр по вертикали }
 Cntr.Y: = Dim.Y div 2;                  { Центр по горизонтали }
 I: = 1; J: = 1;
 repeat
```

```
    if i < Cntr.X then
                                    { "Раздвигание" слайда по вертикали }
    begin
      if j = Cntr.Y then
      begin
        CopyBlock(0,0,Loc.X,Loc.Y,Dim.X,Dim.Y);
        i: = Cntr.X;
      end
      else
      begin
        CopyBlock(Cntr.X-i,0,Loc.X+Cntr.X-i,Loc.Y,2*i,Dim.Y);
        Inc(i)
      end;
    end;
    if j < Cntr.Y then          { "Раздвигание" слайда по горизонтали }
    begin
      if i = Cntr.X then
      begin
        CopyBlock(0,0,Loc.X,Loc.Y,Dim.X,Dim.Y);
        j: = Cntr.Y;
      end
      else
      begin
        CopyBlock(0,Cntr.Y-j,Loc.X,Loc.Y+Cntr.Y-j,Dim.X,2*j);
        Inc(j)
      end;
    end;
    Delay(CurrentDelay);      { Задержка в CurrentDelay, мс }
  until (i = Cntr.X) and (j = Cntr.Y);
end;
```

Листинг 12. Эффект Diagonal

```
Diagonal: begin
 j: = 2 + (Dim.X div Grain) + (Dim.Y div Grain); { Количество диагоналей }
 i: = 1;
 Cntr.X: = 0;
 Cntr.Y: = 0;
 CopyGrain(Cntr,Loc,Dim.X,Dim.Y);
 While i < = j do
 begin
    for Cntr.X: = i downto 0 do
    begin                                   { Копирование ячеек, }
      Cntr.Y: = i-Cntr.X;                   { расположенных на }
      CopyGrain(Cntr,Loc,Dim.X,Dim.Y);      { i-й диагонали слайда }
    end;
    Inc(i);
    Delay(CurrentDelay);              { Задержка в CurrentDelay, мс }
 end;
end;
```

Листинг 13. Эффект Spiral

```
Spiral: begin
 j: = Dim.X div Grain;              { Количество "строк" и "столбцов" ячеек }
 i: = Dim.Y div Grain;              { для рассматриваемого слайда }
 Cntr.X: = j div 2;                   { Центральный "столбец" }
 Cntr.Y: = i div 2;                   { Центральная "строка" }
 CopyGrain(Cntr,Loc,Dim.X,Dim.Y); { Копирование центральной ячейки }
 j1: = 0;
 i1: = 0;
 repeat
    Inc(i1,2);                      { Переход на следующий виток спирали }
    Inc(j1,2);
    Inc(Cntr.Y);
    CopyGrain(Cntr,Loc,Dim.X,Dim.Y);
    for K: = 2 to j1 do
    begin
      Inc(Cntr.X);                  { Копирование ячеек вправо от текущей }
      CopyGrain(Cntr,Loc,Dim.X,Dim.Y);
    end;
    Delay(CurrentDelay);            { Задержка в CurrentDelay, мс }
    for K: = 1 to i1 do
    begin
      Dec(Cntr.Y);                  { Копирование ячеек вверх от текущей }
      CopyGrain(Cntr,Loc,Dim.X,Dim.Y);
    end;
    Delay(CurrentDelay);            { Задержка в CurrentDelay, мс }
    for K: = 1 to j1 do
    begin
      Dec(Cntr.X);                  { Копирование ячеек влево от текущей }
      CopyGrain(Cntr,Loc,Dim.X,Dim.Y);
    end;
    Delay(CurrentDelay);            { Задержка в CurrentDelay, мс }
    for K: = 1 to i1 do
    begin
      Inc(Cntr.Y);                  { Копирование ячеек вниз от текущей }
      CopyGrain(Cntr,Loc,Dim.X,Dim.Y);
    end;
    Delay(CurrentDelay);            { Задержка в CurrentDelay, мс }
  until (j1 > j) and (i1 > i);
end;
```

Листинг 14. Подготовительная процедура для эффекта Random

```
procedure PrepearRand(var RandomArea; SX,SY: integer;
                      var k,j: word);
var
 i, ti, tii: word;
 Ra: array [0..8959] of word absolute RandomArea;
begin
 repeat              { Подсчет общего числа ячеек в слайде }
    k: = (Sx div Grain)*(Sy div Grain)-1;
    Inc(Grain);
 until k < 8959;    { Если число ячеек больше 8959, увеличиваем ячейку }
 Dec(Grain);
 j: = (Sx div Grain);
 randomize;
            { Элементам массива Ra присваиваем номера ячеек с 0 по k }
 for i: = 0 to k do Ra[i]: = i;
 for i: = 0 to 2*k do
 begin
    ti: = System.Random(k);    {Вполне приемлемая последовательность }
    tii: = Ra[1];              {получается после 2*k случайных }
    Ra[1]: = Ra[ti];           {перестановок элементов массива Ra. }
    Ra[ti]: = tii;
 end;
end;
```

Листинг 15. Эффект Random

```
Random : begin
 { Подготовка псевдослучайной последовательности выводимых ячеек }
 PrepearRand(RandomArea,Dim.X,Dim.Y,i1,j1);
 for i: = 0 to i1 do
 begin
    Cntr.Y: = RandomArea[i] div j1;          { Пересчет номера ячейки }
    Cntr.X: = RandomArea[i] - (Cntr.Y*j1);   { в ее координаты }
    CopyGrain(Cntr,Loc,Dim.X,Dim.Y);
    Delay(CurrentDelay);              { Задержка в CurrentDelay, мс }
 end;
end;
```

Листинг 16. Пример использования модуля SlideEffects

```
uses
 Graph, SlideEffects, Objects, Crt;

var
 grDriver: Integer;
 grMode: Integer;
 ErrCode: Integer;
 i: Effects;
 Loc: TPoint;

begin
 grDriver : = EGA;
 grMode : = EGAHi;
 { Инициализация графического режима }
 InitGraph(grDriver, grMode,'');
 ErrCode : = GraphResult;
 if ErrCode = grOk then
 begin
 { Инициализация эффектов }
    EffectsInit;
    Loc.X: = 0;
    Loc.Y: = 0;
 { Вывод на экран слайда из файла SLIDE.PCX
 с применением эффектов }
    for i: = Wipe to Strips do
    begin
      CurrentEffect: = i;
      ShowWithEffect('SLIDE.PCX',Loc);
      Delay(5000);
      ClearDevice;
    end;
    Readln;
    CloseGraph;
 { Возврат из графического режима }
 end
 else Writeln('Graphics error:', GraphErrorMsg(ErrCode));
end.
```

Стандарт PCX + эффекты

С. Курковский

Это статья для тех, кто хочет сэкономить 99$.

Сначала несколько слов о том, что стало причиной появления данной работы. Одним из наиболее распространенных форматов кодирования графических образов является формат PCX, который предложила фирма ZSoft для редактора PC PaintBrush. Этот формат обладает очень многими достоинствами: прост в кодировании, им пользуются многие прикладные пакеты, и, главное, для работы с изображениями в этом формате уже написано много функций и библиотек. Среди них — библиотека PCX Programmer's Toolkit, созданная фирмой Genus Microprogramming. Эта библиотека (автор пользовался версией 3.53) предоставляет программисту функции всевозможного назначения, с их помощью можно написать очень многое: от собственной программы обработки изображений до диалогового архиватора картинок. Кроме функций, примеров их использования и прикладных программ, в PCX Programmer's Toolkit входит демонстрационная программа FXDEMO, рекламирующая библиотеку эффектов (эффектами здесь называются различные методы появления изображений на экране). На первый взгляд, эффекты были довольно сложными: картинка «выезжала» на экран с разных сторон, заполнялась квадратиками по диагонали, расширялась из центра экрана до полного размера, и еще многое-многое другое. После того как программист был сражен наповал поразившими его эффектами, ему предлагалось приобрести в личное пользование библиотеку ни много ни мало, а всего лишь за 99$. Но при детальном рассмотрении эффекты оказались не такими уж и сложными, и появилась идея написать такую библиотеку самому. Приводимые здесь функции не являются полным аналогом библиотеки эффектов Genus. Во-первых, Chris Howard (автор библиотеки) писал на ассемблере, а не на C++. Во-вторых, приводимая библиотека по количеству эффектов меньше, но новые эффекты конструируются очень просто подбором нового алгоритма в 5-10 строк. В-третьих, данная библиотека содержит несколько дополнительных эффектов, которых не было у Genus Microprogramming.

Ядром библиотеки является класс pcxImage (листинг 1), который обеспечивает чтение картинки из файла, ее хранение и предоставляет несколько функций вывода картинки на экран. Самая простая — pcxImage::ShowImage. Эта функция просто показывает картинку целиком. Функция pcxImage::ShowLine показывает одну строчку картинки высотой в один пиксел, с помощью этой функции реализованы дополнительные эффекты. И, наконец, функция pcxImage::ShowBox обеспечивает вывод на экран прямоугольника размером 8x8 для EGA или 8x11 для VGA. С помощью этой функции работает большинство эффектов. Все описанные функции вызывают функцию pcxVirtualDisplay из библиотеки PCX Programmer's Toolkit. Эта функция отображает на экране прямоугольную часть графического образа, предварительно загруженного в память. Безусловно, можно было бы написать эту функцию самому и привести ее здесь, но это не сделано по двум причинам: во-первых, из соображений экономии места и, во-вторых, потому что нечто подобное уже было сделано в работе А. Тихонова «Показать картинку...», опубликованной в «Мониторе» № 3 за 1992 год.

Описания констант (pcxSUCCESS и так далее), прототипов библиотечных функций содержатся в PCXLIB.H. Этот файл включен в PCX Programmer's Toolkit и здесь, естественно, не приводится.

Класс FX (листинг 2), наследующий pcxImage, содержит алгоритмы работы эффектов, каждый из которых представляет собой некоторую последовательность вызовов функции ShowLine или ShowBox. Теперь расскажем несколько подробнее о работе функций класса FX.

Изображение, на котором демонстрируются эффекты, приведено на рис. 1.

Рис. 1

Эффект FX::FillBoxes выводит картинку на экран, начиная с одного из углов изображения, заполняя его прямоугольниками по диагонали (рис. 2). Задав направление заполнения, можно получить четыре метода появления картинки. Во внешнем цикле переменная Count принимает значение текущего размера диагонали квадратной области, целиком вмещающей в себя картинку. Переменные XBox и YBox — текущие координаты прямоугольника, который будет показан.

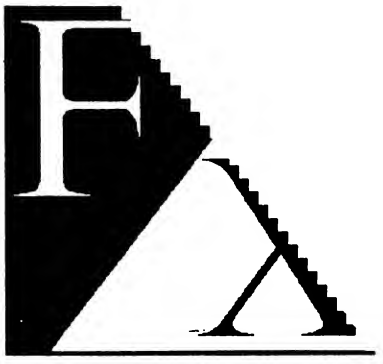


Рис. 2

Эффект FX::RandomBoxes заполняет картинку случайно появляющимися прямоугольниками (рис. 3). Нетривиальный момент в этой функции заключается в отыскании следующего прямоугольника, который надо показать. Для этой цели создается вектор Boxes, в котором каждому прямоугольнику соответствует один бит. На каждом шаге берется случайное число CurrBox и, если в Boxes на соответствующем месте стоит 1 (данный прямоугольник уже показан), операция повторяется. В противном случае в соответствующий бит записывается 1, делаются необходимые проверки на выход за границы изображения и вызывается функция ShowBox. У данного подхода есть один недостаток: при значительных размерах картинки в конце работы данного эффекта время для отыскания следующего прямоугольника может несколько увеличиться по сравнению с началом работы, но это компенсируется минимальными затратами памяти.

Рис. 3

Эффект FX::PullLines работает следующим образом: вся картинка делится на несколько горизонтальных областей одинаковой высоты, и все области прорисовываются одновременно (рис. 4). Прорисовка может начинаться сверху или снизу, таким образом одна функция даст нам два метода, которых, заметим, не было в библиотеке PCX Programmer's Toolkit. Этот эффект работает настолько быстро, что приходится вводить задержку во времени внутри главного цикла, чтобы было заметно, как появляется картинка. Этот эффект очень хорошо подходит для больших картинок, например, заставок.

Рис. 4

Эффект FX::LineSpread подходит в основном для черно-белых изображений (рис. 5). Идея предельно проста: каждая строка картинки прорисовывается, начиная с самой верхней строки, до своего истинного расположения.

Рис. 5

Эти четыре эффекта дают направление дальнейших действий по написанию новых, более оригинальных и сложных эффектов, которые обязательно должны появиться в Ваших собственных программах. ■

Листинг 1.
Описание класса pcxImage

```
#ifndef _PCXIMAGE_
#define _PCXIMAGE_

#include "pcxlib.h"

class pcxImage
{
public:
    int   Height;                       // Размеры картинки
    int   Width;                        // в пикселах
    int   BoxHeight;                    // Высота прямоугольника
    int   BHeight;                      // Размер картинки
    int   BWidth;                       // в прямоугольниках
    int   ErrorCode;                    // Код ошибки
    long BufSize;                       // Размер буфера картинки
    long PicBuf;                        // Адрес буфера картинки

    pcxImage(char *PicFiLename, int VideoMode);   // Конструктор
    ~pcxImage();                                  // Деструктор

    // Различные функции вывода изображений
    int ShowImage(int x, int y, int videopage);
    int ShowBox(int xbox, int ybox, int x, int y, int videopage);
    int ShowLine(int line, int x, int y, int videopage);
};
#endif
```

Определение класса pcxImage

```
#include "pcxImage.hpp"
#include <stdlib.h>

pcxImage::pcxImage(char *PicFileName, int VideoMode)
{
    BufSize = Width = Height = 0;
    PicBuf = 0;
    pcxSetDisplay(VideoMode);

    if((ErrorCode = pcxVerifyFile(PicFileName))! = pcxSUCCESS) return;

    PCXHEADER h;
    if((ErrorCode = pcxGetFileHeader(PicFileName,&h))! = pcxSUCCESS)
        return;

    Width = h.x2 - h.x1 + 1;
    Height = h.y2 - h.y1 + 1;
    if(Width % 8 > = 4) Width = (Width/8+1)*8;
        else Width = Width/8*8;

    if(VideoMode < = pcxEGA_10) BoxHeight = 8;
    else BoxHeight = 11;

    BHeight = Height/BoxHeight+1;
    BWidth = Width/8+1;

    BufSize = pcxVirtualSize(VideoMode,Width,Height);

    if((PicBuf = (long) new char[BufSize]) = = NULL) return;
    if((ErrorCode = pcxFileVirtual(PicFileName,PicBuf))! = pcxSUCCESS)
        return;
}

pcxImage::~pcxImage()
{
    delete &PicBuf;
}

int pcxImage::ShowImage(int x, int y, int videopage)
{
    return ErrorCode = pcxVirtualDisplay(PicBuf,0,0,x,y,x+Width-1,
                                         y+Height-1,videopage);
}

int pcxImage::ShowBox(int xbox, int ybox, int x, int y, int videopage)
{
    int Dx = min(8,Width-xbox*8-1);
    int Dy = min(BoxHeight, Height-ybox*BoxHeight-1);
    if(Dx < = 0 | | Dy < = 0) return (ErrorCode = pcxSUCCESS);
    return ErrorCode = pcxVirtualDisplay(PicBuf, xbox*8, ybox*BoxHeight,
                        x+xbox*8, y+ybox*BoxHeight,
                        x+xbox*8+Dx, y+ybox*BoxHeight+Dy, videopage);
}

int pcxImage::ShowLine(int line, int x, int y, int videopage)
{
    return ErrorCode = pcxVirtualDisplay(PicBuf, 0, line,
                        x, y+line, x+Width-1, y+line, videopage);
}
```

Листинг 2.
Описание класса FX

```
#ifndef _FX_
#define _FX_

#include "pcxImage.hpp"

enum VertDirection {Up, Down};
enum HorizDirection {Left, Right};

class FX: public pcxImage
{
public:
    FX(char *FileName, int VMode);
    FX(char *LibFileName, char *FileName, int VMode);

    int FillBoxes(int x, int y, VertDirection, HorizDirection);
    int PullLines(int x, int y, int LineHeight, VertDirection, int Delay);
    int LineSpread(int x, int y);
    int RandomBoxes(int x, int y);
};
#endif
```

Определение класса FX

```
#include "FX.hpp"
#include <stdlib.h>
#include <dos.h>        // delay
#include <mem.h>        // setmem

FX::FX(char *file, int vmode):pcxImage(file,vmode){}

int FX::FillBoxes(int x, int y, VertDirection VDir, HorizDirection HDir) {
    for(int Count = 0; Count < BHeight*BWidth; Count++) {
        for(int i = 0; i < = Count ; i++) {
            unsigned XBox = i;
            unsigned YBox = Count - XBox;
            if(XBox > = BWidth | | YBox > = BHeight) continue;

            if(HDir = = Left)     XBox = BWidth -XBox-1;
            if(VDir = = Up)       YBox = BHeight-YBox-1;
            int RetValue = ShowBox(XBox, YBox, x, y, 0);
            if(RetValue < 0) return RetValue;
        }
    }
    return pcxSUCCESS;
}

int FX::RandomBoxes(int x, int y) {
    unsigned XBox, YBox;
    unsigned CurrBox, MaxFill = BHeight*BWidth;
    char *Boxes;

    if((Boxes = new char[MaxFill+1]) = = NULL) return pcxERR_CMMFAIL;
    memset(Boxes, 0, MaxFill);

    for(int Filled = 0; Filled < = MaxFill; Filled++) {
        do{
            CurrBox = random(MaxFill);
        }while(Boxes[CurrBox > > 3] & (1 < < (CurrBox % 8)));

        Boxes[CurrBox > > 3] | = 1 < < (CurrBox % 8);
        if((XBox = CurrBox % BWidth) > BWidth) continue;
        if((YBox = CurrBox / BWidth) > BHeight) continue;

        int RetValue = ShowBox(XBox, YBox, x, y, 0);
        if(RetValue < 0) {
            delete Boxes;
            return RetValue;
        }
    }
    delete Boxes;
    return pcxSUCCESS;
}

int FX::PullLines(int x, int y, int LineSize,
        VertDirection Direction, int Delay) {
    int First = 0, Limit = 0, Delta;

    switch(Direction)   {
        case Down: Limit = LineSize; Delta = 1; break;
        case Up: First = LineSize; Delta = -1; break;
    }

    for(int CurrSize = First; CurrSize*Delta < Limit*Delta; CurrSize+ = Delta)
        {
        for(int CurrLine = 0; CurrLine < Height ; CurrLine+ = LineSize) {

            int ActualSize = max(0, CurrLine+CurrSize-1);
            if(ActualSize > = Height) continue;

            int RetValue = ShowLine(ActualSize, x, y, 0);
            if(RetValue < 0) return RetValue;
        }
        delay(Delay);
    }
    return pcxSUCCESS;
}

int FX::LineSpread(int x, int y) {
    for(int CurrentLine = Height-1; CurrentLine > = 0; CurrentLine--) {
        for(int CurrentY = CurrentLine; CurrentY > = 0; CurrentY--) {
            int RetValue = ShowLine(CurrentLine, x, y-CurrentY, 0);
            if(RetValue < 0) return RetValue;
        }
    }
    return pcxSUCCESS;
}
```

ЛАНИТ NOVELL

"КЛИЕНТ-СЕРВЕР" СУБД : РЕШЕНИЯ NOVELL И ЛАНИТ

NetWare Btrieve

Менеджер записей, реализованный как сервер БД низкого (по уровню запросов к данным) уровня.

Файлы данных с записями фиксированной и переменной длины, ключи различных типов, доступ к записям как в логическом, так и в физическом порядке, реализует групповые (Extended) запросы к БД, допускает сжатие данных и их шифрацию, обеспечивает парольную защиту, механизм блокировки, транзакций, восстановления (Roll Back) и прокрутки (Roll Forward) и т.д.

NetWare SQL v.3.0

Реализует в сети NetWare SQL - сервер баз данных.

Загружаемый модуль NetWare (NLM), работающий на файловом сервере и осуществляющий интерфейс между БД на дисковой системе сервера и приложениями на рабочих станциях на основе языка запросов SQL.

ЛАНИТ предлагает также ряд программных продуктов собственной разработки для создания приложений для работы с NetWare Btrieve.

Lanit Developer v.1.1

Средство проектирования, использующее простой язык описания разрабатываемой информационной системы.

LanitDeveloper использовался для создания ряда приложений: LD_Office - для коммерческой фирмы, RestauLAN - для ресторана с возможностью расчетов по магнитным картам и др.

Lanit Engine v.1.03

Обеспечивает работу с Btrieve как с реляционной базой данных с использованием SQL - ориентированного интерфейса.

Обеспечивает создание приложений на языках C, Asm, Pascal, Basic, Cobol.

Clipper Interface for LanitEngine

Библиотека для работы с LanitEngine из среды Clipper.

Включает также набор макроопределений для препроцессора Clipper и интерактивную Clipper-утилиту работы с Btrieve базой данных.

ЛАНИТ поставляет эти продукты, консультирует по использованию, разрабатывает приложения по заказу.

107066, Москва, ул.Доброслободская, 5

Тел. (095)261-4234, 261-1870, 261-5781 (факс)

Современные модемы для современных телекоммуникаций

Не нужно доказывать, что информатизация требует развития телекоммуникаций.
Не нужно доказывать, что современные телекоммуникации требуют использования качественных модемов.
Мы много лет потратили на развитие телекоммуникаций в нашей стране, и сейчас мы предлагаем только лучшее. Суперсовременные модемы ZyXEL, Telebit, US/Robotics, универсальные модемы Discovery, выдержавшие проверку временем, любые компьютеры для телекоммуникационных систем.

Discovery 2400CM/D — внешний модем для надежных телекоммуникационных систем общего назначения. Аппаратная реализация MNP5 и V.42bis. Отправка факсов. Сертификат Минсвязи.

Discovery 2496HX — встраиваемый факс-модем, отлично подходящий для российского офиса. Прием и передача факсов, аппаратная реализация MNP5 и V.42bis.

Discovery 2496PX — карманный факс-модем для тех, кто работает всегда и везде. MNP5, V.42bis, питание от батареи 9В или от сети.

Telebit Worldblazer — скоростной факс-модем (до 70,000 бит/с), прославившийся своей надежностью.

ZyXEL U-1496 — высокоскоростные модемы (до 57,600бит/с) с возможностью передачи и приема факсов и речевых сообщений в цифровой форме.

demos™

Телефоны: (095) 231-21-29, 231-63-95, 233-05-92, 233-02-42, 231-60-02
Факс: (095) 233-50-16
E-mail: info@hq.demos.su

Знак и логотип demos — торговая марка фирмы Демос

Резидентные программы на языке С

Д. Валдин

Часть 2

Итак, в первой части статьи мы научились создавать резидентные программы на языке С. Однако мы остановились на пессимистической ноте, перечислив ряд серьезных погрешностей в написанной нами программе. Давайте зададимся целью устранить все перечисленные недостатки без увеличения размера EXE файла. При этом напишем программу таким образом, чтобы ее можно было компилировать как в среде **Microsoft**, так и в среде **Borland**. Начнем по порядку.

Предотвращение повторной загрузки

Вполне естественно, что хорошая TSR программа должна уметь самостоятельно определять наличие в памяти своей резидентной копии и не допускать повторной загрузки. Это значит, что, прежде чем остаться резидентной, программа должна каким-то уникальным способом обозначить свое присутствие в памяти. Посмотрим, как решают эту задачу профессиональные резидентные программы.

Norton Guide для определения того, был ли он ранее загружен в память, помещает в регистр AX «магическое число» F398H и вызывает прерывание 16H. Прерывание **BIOS** 16H содержит набор функций для работы с клавиатурой. Номер функции задается содержимым регистра AH; функция с номером F3H отсутствует. **Norton Guide**, оставаясь резидентным, перехватывает это прерывание. Его обработчик предусматривает, что если регистр AX содержит F398H, то это значение заменяется другим «магическим числом» — 6A73H, после чего осуществляется немедленный выход из прерывания. Таким образом, если после вызова прерывания в регистре AX находится 6A73H, то **Norton Guide** уже загружен, а если любое другое значение, то его можно загружать. Правда, существует вероятность, что какая-нибудь другая программа воспользуется тем же прерыванием с тем же набором «магических чисел» [1].

Turbo Help фирмы **Borland** поступает по-другому. Он использует так называемое мультиплексное прерывание INT 2FH, которое специально предназначено для взаимодействия с резидентными процессами самой **DOS**, такими как **PRINT**, **ASSIGN** и **SHARE**. **THELP** перехватывает это прерывание и назначает себе уникальный мультиплексный номер. Присутствие в памяти процесса с этим номером и является критерием того, что **THELP** уже установлен.

Фирма **Blaise Computing** в своем известном инструментарии **Blaise C Tools** предлагает следующий подход. Текст каждого обработчика прерывания написан таким образом, что в самом его начале резервируется место для хранения указателя на управляющий блок TSR программы, содержащий сигнатуру, определяемую пользователем. Считается, что программа уже является резидентной, если хотя бы один из обработчиков прерываний, перехваченных ею, содержит в своем теле указатель на этот управляющий блок. Нетрудно догадаться, однако, что этот метод потерпит фиаско в случае, если все векторы прерываний, перехваченных программой, будут изменены другим резидентным процессом.

Существует еще один способ решения вышеупомянутой проблемы. Суть его заключается в следующем. Перед тем как остаться резидент-

ной, программа записывает в определенное место PSP уникальную сигнатуру. При попытке повторного запуска программа проверяет наличие этой сигнатуры в PSP всех загруженных в память программ. Если она найдена, то считается, что программа уже резидентна и необходимость в повторной установке отпадает. Чтобы реализовать описанный алгоритм, необходимо представлять себе, как операционная система осуществляет контроль за распределением памяти.

Распределение памяти

Вся рабочая память, доступная MS-DOS, поделена на блоки. Отдельные блоки памяти организованы в цепочку. В начале каждого блока находится служебное поле длиной в 16 байтов, где записаны размер блока и индикация, является ли данный блок занятым или свободным. Это служебное поле называется управляющим блоком памяти (MCB — Memory Control Block) (табл. 2).

Таблица 2

Формат управляющего блока памяти

Смещение	Длина	Содержимое
+0	1	90 (5AH), если блок является последним в цепочке; в противном случае — 77 (4DH);
+1	2	0, если блок свободен; сегментный адрес PSP программы-владельца, если блок занят;
+3	2	размер блока в параграфах (по 16 байтов);
+5	3	зарезервировано (не используется);
+8	8	имя программы в формате ASCIIZ, если блок содержит PSP программы-владельца; мусор — в противном случае (это поле поддерживается, начиная с версии MS-DOS 4.0, и в более ранних версиях не используется).

На языке C MCB может быть представлен такой структурой (пример 10).

Пример 10

```
#pragma pack(1)
typedef struct {
  char      ident;
  unsigned  owner_psp;
  unsigned  size;
  char      _reserved[3];
  char      name[8];
} MCB;
#pragma pack()
```

Обратите внимание на директиву #pragma, предшествующую описанию структуры MCB. Ее присутствие обязательно, поскольку по умолчанию компилятор выравнивает все поля структуры на границу слова, что для нас здесь совершенно неприемлемо. Ведь в этом случае поле owner_psp будет располагаться со смещением +2 относительно начала структуры, а не +1, как это есть на самом деле. Чтобы заставить компилятор осуществлять выравнивание на границу байта, необходимо включить в текст программы директиву #pragma pack(1). Восстановить нарушенное умолчание можно директивой #pragma pack().

Адрес первого MCB получим с помощью недокументированной функции DOS 52H (Get List of Lists). Сделаем это следующим образом (пример 11).

Пример 11

```
unsigned mcbseg;

_asm  {
  push es
  mov ah, 52h
  int  21h
  mov ax, es:[bx-2]
  mov mcbseg, ax
  pop  es
}
```

После выполнения этого кода переменная mcbseg будет содержать сегментный адрес первого MCB (смещение равно 0). Просмотрим теперь всю цепочку MCB в поисках PSP, содержащего нашу уникальную сигнатуру (пример 12).

Пример 12

```
#define IV_NOT_FOUND 0
#define IV_INSTALLED  1

static char *IV_SIGNATURE = "Alarm 1.0";

  ...
  while ( 1 ) {
      MCB _far *pmcb = MK_FP( mcbseg, 0 );

      if ( pmcb->owner_psp = = mcbseg + 1 ) {
          char _far *signature = MK_FP(  pmcb->owner_psp,
                                         SIGN_OFFSET);

          if ( _fstrcmp( signature, IV_SIGNATURE ) = = 0 )
              return IV_INSTALLED;

      }
      if ( pmcb->>ident = = 0x4d )
          mcbseg + = pmcb->size + 1;
      else
          break;
  }
  return IV_NOT_FOUND;
```

В описанном алгоритме осталось лишь одно «белое пятно». Мы еще не решили, в какое место PSP будем записывать нашу сигнатуру. Иными словами, нам предстоит определить, чему равно SIGN_OFFSET. Посмотрим на структуру PSP (табл. 3).

Таблица 3

Структура полей PSP

Смещение	Длина	Содержание
0 (00H)	2	Инструкция INT 20H (завершение программы)
2 (02H)	2	Размер памяти в параграфах
4 (04H)	1	Зарезервировано
5 (05H)	5	Код операции вызова диспетчера функций DOS
10 (0AH)	4	Адрес завершения (вектор 22H)
14 (0EH)	4	Адрес обработки Ctrl-Break (вектор 23H)
18 (12H)	4	Адрес обработки ошибок (вектор 24H)
22 (16H)	22	Зарезервировано
44 (2CH)	2	Адрес среды
46 (2EH)	40	Зарезервировано
85 (5CH)	16	Первый параметр командной строки
101 (6CH)	16	Второй параметр командной строки
128 (80H)	128	Командная строка / Область обмена с диском

Содержимое PSP отражает этапы развития операционной системы. При реализации некоторых новых возможностей в PSP появляются новые поля. С другой стороны, поля, которые широко использовались раньше, в настоящее время не представляют практически никакого интереса. Одно из таких полей расположено со смещением +5CH относительно начала PSP. Оно может быть полезно программам, использующим для работы с файлами метод FCB. Однако, поскольку этот метод считается устаревшим, поле можно использовать для других целей, например, для хранения сигнатуры программы, как в нашем случае. Самым важным здесь является то, что ни одна функция из C Run-Time Library не претендует на эту область памяти. Таким образом, мы можем быть уверены, что записанная нами сигнатура сохранится там на всем протяжении времени, пока программа будет оставаться резидентной.

Удаление программы из памяти

На первый взгляд, удалить программу из памяти не так уж сложно. Необходимо просто восстановить векторы прерываний, перехваченных нашей программой, а затем освободить память, занимаемую ею. Однако, прежде чем решиться на подобную акцию, мы должны удостовериться в том, что наша программа по-прежнему владеет всеми прерываниями, которые она перехватила. В противном случае мы рискуем «подложить свинью» той резидентной программе, которая была загружена после нашей. Ведь если эта программа написана корректно, то, очевидно, перед тем как изменить вектор прерывания, она сохранила его значение в своем теле и обращается по этому вектору в процессе своей работы, передавая тем самым управление нашему обработчику прерывания. Если мы выгрузим нашу программу из памяти, на ее место будет загружена какая-нибудь другая программа. Управление, между тем, будет, как и прежде, передаваться в то место, где когда-то находился наш обработчик прерывания. Полагаю, что такие действия вряд ли приемлемы для операционной системы. Значит, прежде чем удалить программу из памяти, необходимо проверить, все ли векторы прерываний, перехваченных ею, указывают на ее собственные обработчики прерываний. Чтобы сделать это, нерезидентная копия программы должна обладать информацией об адресах процедур прерываний. Получается, что программа перед обращением к функции **_dos_keep()** должна где-то сохранить не только свою сигнатуру, но и значения векторов прерываний как до момента перехвата, так и после него.

Да, пожалуй, всю эту информацию в PSP не запишешь — места не хватит. Поэтому давайте поместим в PSP только две ссылки: ссылку на сигнатуру и ссылку на массив векторов перехваченных прерываний. Заметим, что, поскольку сигнатура и массив могут располагаться в любом месте памяти, ссылки должны быть типа _far. В принципе, можно описать эти ссылки как близкие (типа _near), однако тогда потребуется сохранить в PSP еще и значение сегмента данных TSR программы. В этом случае нерезидентная копия программы сможет сформировать правильные адреса, комбинируя указанные сегмент и смещения. Однако давайте не будем усложнять себе жизнь и возложим на компилятор все заботы, связанные с сегментацией памяти. Объединим теперь эти ссылки в единую структуру (пример 13) и назовем ее управляющим блоком TSR программы **(ICB — Intervention Control Block)**.

Пример 13

```
typedef struct {
  char      _far *signature;
  IVECT     _far *pvect;
} ICB;
```

Объявим тип IVECT (пример 14) как:

Пример 14

```
typedef struct {
  int           num;
  OLD_VECT      old;
  NEW_VECT      new;
} IVECT;
```

В свою очередь, объявления типов **OLD_VECT** и **NEW_VECT** запишем следующим образом (пример 15).

Пример 15

```
typedef void ( _interrupt _far * _far * OLD_VECT) ();
typedef void ( _interrupt _far * NEW_VECT) ();
```

Таких, на первый взгляд, сложных объявлений не стоит пугаться. Достаточно лишь пару раз попробовать записать их самостоятельно, и все будет понятно. А зато каким изящным становится определение массива vect (пример 16).

Пример 16

```
IVECT vect[] = {
   { 0x08, &oldtimer, newtimer },
   { 0x10, &oldvideo, newvideo },
};
```

При таком подходе алгоритм получения значений старых векторов прерываний и установления новых векторов не зависит от количества прерываний, перехватываемых программой (пример 17).

Пример 17

```
#define NVECTS (sizeof( vect ) / sizeof( vect[0] ))

  register int i;

  for   ( i = 0; i < NVECTS; i++ ) {
        /* Получить адрес старого обработчика прерывания. */
        *(vect[i].old) = _dos_getvect( vect[i].num );

        /* Установить собственный обработчик. */
        _dos_setvect( vect[i].num, vect[i].new );

  }
```

С учетом нововведений изменим слегка алгоритм определения присутствия в памяти резидентной копии нашей программы (пример 18).

Пример 18

```
#define ICB_OFFSET 0x5c

ICB _far *picb;        // Переменная picb должна быть глобальной.

...
if ( pmcb->owner_psp == mcbseg + 1 ) {
     picb = MK_FP( pmcb->owner_psp, ICB_OFFSET );

     if ( _fstrcmp( picb->signature, IV_SIGNATURE ) == 0 )
          return IV_INSTALLED;
}
...
```

Теперь мы, наконец, располагаем информацией, достаточной для того, чтобы написать процедуру удаления резидентной программы из памяти. Процесс восстановления старых векторов прерываний можно реализовать следующим образом (пример 19).

Пример 19

```
register int i;

for ( i = 0; i < NVECTS; i++ )
  _dos_setvect( picb->pvect[i].num, *(picb->pvect[i].old ));
```

Алгоритм освобождения памяти, по существу, аналогичен тому, каким мы пользовались при определении наличия в оперативной памяти резидентной копии нашей программы. Условие, проверяющееся для каждого MCB, должно, конечно же, быть другим (пример 20).

Пример 20

```
if ( pmcb->owner_psp == FP_SEG( picb ))
  _dos_freemem( mcbseg + 1 );
```

Тексты функций **iv_sense()**, **iv_install()**, **iv_remove()** приведены в листинге 6, поэтому не будем больше заострять внимание на реализации отдельных фрагментов и перейдем к обсуждению следующего вопроса.

Передача данных резидентной программе

В каких случаях может возникнуть необходимость в передаче данных резидентной программе? Представьте себе, что мы пользуемся улучшенной версией программы **ALARM** и поставили будильник на 12 часов. Он прозвенел, и мы хотим переставить его, скажем, на 16 часов. Запускаем программу, указав в командной строке новое время. И что мы видим: программа мрачно сообщает нам, что она уже загружена в память, и отказывается воспринимать новую установку времени. А чего же мы, собственно говоря, хотим? Ведь мы сами запрограммировали ее таким образом. Как же нам поступить? Конечно, можно выгрузить программу из памяти, благо мы уже научились это делать, а затем снова сделать ее резидентной. Интересно, а может быть можно организовать работу таким образом, чтобы появилась возможность редактирования времени без необходимости выгрузки резидентной программы из памяти? Почему бы и нет. Сделаем это так. Определим тип **IVDATA**, заведем переменную этого типа и поместим в нее все те параметры, которые определяют начальную установку программы. В нашем случае это две переменные: **ftimesup** и **goaltick** (пример 21).

Пример 21

```
typedef struct {
  long goaltick;
  int  ftimesup;
} IVDATA;

IVDATA ivdata;
```

Добавим в структуру ICB еще одно поле. Теперь управляющий блок TSR программы будет выглядеть так (пример 22).

Пример 22

```
typedef struct {
  char     _far *signature;
  IVECT    _far *pvect;
  IVDATA   _far *ivdata;
} ICB;
```

Поля переменной **ivdata** заполним в функции **main()**. Теперь, если функция **iv_sense()** сообщит нам, что в памяти уже есть резидентная копия нашей программы, то мы не будем торопиться выдавать на консоль грозное сообщение. Вместо этого вызовем функцию **iv_update()**, которая скопирует значения всех полей переменной **ivdata** в соответствующее место памяти, где находится переменная **ivdata** резидентной копии программы (пример 23).

Пример 23

```
void iv_update( void )
{
  *(picb->ivdata) = ivdata;
}
```

Цель достигнута: резидентная копия нашей программы получила новую установку времени.

На вопросе разбора командной строки и проверки устанавливаемого времени на допустимость введенного значения останавливаться не будем — это дело техники, а вот о том, как уменьшить размер EXE файла TSR программы, поговорить стоит.

Способы уменьшения исполняемого кода резидентной программы

Первое, что приходит на ум, — обработать исполняемый файл каким-нибудь упаковщиком — в нашем случае совершенно не подходит. Да, EXE файл, без-

условно, уменьшится, однако при загрузке его в память исполняемый код не только восстановится до первоначальных размеров, но и, скорее всего, увеличится за счет добавления к программе кода, отвечающего за распаковку. Значит, необходимо искать какие-то внутренние резервы. Вот какие рекомендации здесь можно дать.

Во-первых, следует отказаться от всех функций буферизованного вывода (**puts()**, **putchar()** и так далее) и использовать вместо них функции консольного вывода (**cputs()**, **putch()**).

Во-вторых, все операции с файлами следует выполнять на уровне операционной системы, то есть вместо функций **fopen()**, **fclose()**, **fread()**, **fwrite()**, **fseek()** пользоваться функциями **_dos_open()**, **_dos_close()**, **_dos_read()**, **_dos_write()**, **lseek()**. К сожалению, перечисленные выше функции, кроме **lseek()**, отсутствуют в библиотеке **Borland C**, поэтому их придется реализовать самостоятельно, что не так уж трудно, или пользоваться функциями **open()**, **close()**, **read()**, **write()**.

Другой неприятный момент связан с тем, что библиотечная функция **cputs()**, предлагаемая фирмой **Borland**, имеет дополнительные возможности, по сравнению с тезкой из библиотеки **Microsoft**, что сильно сказывается на ее размере. Поэтому ничего не остается, как написать ее «короткую» версию (листинг 4).

Следующий способ уменьшения объема исполняемого кода будет полезен только приверженцам программных продуктов фирмы **Microsoft**. Вы, дорогие друзья, можете еще более уменьшить размер ваших TSR программ, если включите в их тексты функции-пустышки **_setargv()**, **_nullcheck()**, **_fptrap()**, заместив тем самым оригинальные библиотечные функции, которые в резидентных программах, как правило, не используются.

Любителям продукции фирмы **Borland** в утешение могу предложить модифицировать **STARTUP** код, находящийся в файле **C0.ASM**, благо он входит в набор стандартной поставки, исключив из него те места, которые, на Ваш взгляд, являются лишними. Кстати, без модификации этого файла Вам вряд ли удастся написать TSR программу, перехватывающую прерывание **INT 5H** (печать экрана), поскольку **STARTUP** код предусматривает сохранение этого вектора, а также и ряда других перед началом работы программы и восстановление их при ее завершении.

Особенности программной реализации

Итак, модифицированная версия программы **ALARM** написана. Посмотрим на нее повнимательнее. Нетрудно заметить, что в программе присутствует очень много фрагментов, которые могли бы оказаться полезными и в других резидентных программах. В частности, вся рутинная работа, связанная с установкой программы, предотвращением повторной загрузки, передачей параметров резидентной копии и выгрузкой ее из памяти, может быть поручена одному-единственному макросу **install()**. Давайте перенесем разделы программы, которые являются общими для всех TSR программ, в два файла. В файл **VTSR.H** (листинг 5) поместим описания структур, определения констант, макросы и прототипы функций. Файл **VTSR.C** (листинг 6) будет содержать реализацию функций **iv_sense()**, **iv_install()**, **iv_remove()**, **iv_update()** и некоторых других, которые понадобятся нам впоследствии. Еще один файл, **VMISC.H** (листинг 7), будет содержать различные константы и макросы, часто использующиеся в С-программах. Используя условную компиляцию, добьемся того, чтобы программы могли без всяких изменений транслироваться как в среде **Microsoft**, так и **Borland**. Теперь напишем окончательную версию **ALARM.C** (листинг 8), оттранслируем ее различными компиляторами (Batch файлы представлены в листингах 9-12) и затем сравним ее с оригиналом (табл. 4).

Таблица 4

Сравнительные характеристики двух версий программы ALARM

Характеристика	Оригинальная программа	Модифицированная
Контроль повторной загрузки	нет	да
Возможность удаления программы из памяти	нет	да
Возможность редактирования установленного времени	нет	да
Контроль допустимости введенного времени	нет	да
Размер EXE файла / размер резидентной части в байтах		
компилятор Microsoft C 6.0	5805 / 6736	4279 / 4496
Quick C++ 2.51	5933 / 7360	4710 / 4912
Borland C++ 2.0	6626 / 7168	
Turbo C++ 1.0	6668 / 7216	

Программа, которую мы с вами написали сегодня, довольно проста. Зато на ее примере мы смогли, не углубляясь в тонкости реализации процедур прерываний, обсудить проблему написания абстрактной TSR программы. ■

Литература

1. А. Рыскунов. Этот безумный, безумный, безумный мир резидентных программ// КомпьютерПресс.– 1992.– № 4.

Стопроцентное восстановление качества красящих лент для принтеров на специальном оборудовании без извлечения ленты из картриджа.
Тел. (095) 943-25-53

Листинг 4. Файл GETCH.C

```
/* Produce our own versions of putch() and cputs()
 * functions just to reduce program size.
 */
#if defined __TURBOC__
#define _asm  asm
#endif

void putch( int c )
{
    _asm {
        mov ah, 06h
        mov dx, c
        int  21h
    }
}

void cputs( const char *s )
{
    while ( *s )
        putch( *s++ );
}
```

Листинг 5

```
/* Header file: VTSR.H
* Tab stops: 4
* Compilers supported:  QuickC 2.51, Microsoft C 6.0 and later,
*                       Turbo C++ 1.0, Borland C++ 2.0 and later.
* Model: Small
* Author: Dmitri Valdine.
* Email: Relcom (InterNET Department) valdine@ipmo.spb.su
* Last edition: 27-Oct-92.
* Purpose: This file defines the structures, values,
* macros and functions used in TSR-programs.
*/

#ifndef DEF_VTSR
#define DEF_VTSR

#include <dos.h>
#include <stdlib.h>
#include <string.h>
#include <conio.h>

#include "vmisc.h"

#if defined _MSC_VER
    #if (_MSC_VER < 600)
    #error The program should be compiled with Microsoft C 6.0 or later
    #endif
#elif defined __TURBOC__
    #if (__TURBOC__ < 0x295)
    #error The program should be compiled with Turbo C++ 1.0 or later
    #endif
#else
    #error The program should be compiled with Microsoft or Borland compiler
#endif

#if !defined M_I86SM && !defined __SMALL__
    #error The program should be compiled in Small memory model.
#endif

/* Stack and pointer checking off */
#pragma check_stack( off )
#pragma check_pointer( off )

/* Make some substitutions for Borland compilers. */
#if defined __TURBOC__
    #define _interrupt      interrupt
    #define _dos_getvect    getvect
    #define _dos_setvect    setvect
    #define _dos_keep       keep
    #define _dos_freemem    freemem
#endif

/* ICB should be located with the ICB_OFFSET
* from the beginning of PSP.
*/
#define ICB_OFFSET    0x5c

/* Values returned by iv_sence routine */
#define IV_NOT_FOUND      0
#define IV_ATTACHED       1
#define IV_INSTALLED      2

/* Option value for iv_scan routine. */
#define IV_REMOVE         0
#define IV_SENSE          1

/* Define OLD_VECT and NEW_VECT types. */
typedef void ( _interrupt _far * _far * OLD_VECT) ();
typedef void ( _interrupt _far * NEW_VECT) ();

/* Define the IVECT structure */
typedef struct {
    int         num;
    OLD_VECT    old;
    NEW_VECT    new;
} IVECT;

/* Define the intervention control block structure. */
typedef struct {
    char        _far *signature;
    IVECT       _far *pvect;
#if defined _UPDATE_NEED
    void        _far *ivdata;
#endif
} ICB;

#if !defined _MCB_DEFINED

/* Pragma pack argument should be set to 1
* to make byte allignment nesessary for MCB structure.
*/
#pragma pack(1)

/* Define the DOS memory block structure. */
typedef struct {
    char        ident;          // block type: 0x5a if last block in chain,
                                // otherwise 0x4d
    unsigned    owner_psp;      // PSP segment of owner, 0000 if free
                                // 0x0008 if belongs to DOS
    unsigned    size;           // size of memory block in paragraphs
    char        _reserved[3];   // unused
    char        name[8];        // under DOS 4.x + ASCII program name if PSP
                                // memory block, else garbage
                                // null-terminated if less than 8 characters
} MCB;

#pragma pack()

#define _MCB_DEFINED

#endif

/* Install() macro makes easy the TSR-installation process. */
#if defined _UPDATE_NEED
#define if_installed( name ) iv_update(); cputs( #name" updated.\n\r" );
#else
#define if_installed( name ) cputs( #name" already installed.\n\r" );
#endif

#define install( name )                                         \
    if ( remove_it ) {                                          \
        /* Check if our program was already installed. */       \
        switch ( iv_sense() ) {                                 \
            case IV_NOT_FOUND:                                  \
                cputs( #name " not found.\n\r" );               \
                break;                                          \
                                                                \
            case IV_ATTACHED:                                   \
                cputs( "Can't unload until TSRs loaded after "  \
                    #name" are removed.\n\r" );                 \
                break;                                          \
                                                                \
            case IV_INSTALLED:                                  \
                cputs(( iv_remove())                            \
                    ? "Error during removing "#name".\n\r"      \
                    : #name" removed.\n\r" );                   \
                break;                                          \
        }                                                       \
    }                                                           \
    else {                                                      \
        switch ( iv_sense() ) {                                 \
            case IV_INSTALLED:                                  \
            case IV_ATTACHED:                                   \
                if_installed( name )                            \
                break;                                          \
                                                                \
            case IV_NOT_FOUND:                                  \
                cputs( #name" installed.\n\r" );                \
                iv_install();                                   \
                return ERROR;                                   \
        }                                                       \
    }

/* Microsoft and Borland compilers saves registers in differrent ways.
* We should take it into consideration.
*/
#if defined _MSC_VER
#define ARGS unsigned __es,unsigned __ds,unsigned _di,          \
             unsigned _si,  unsigned _bp,  unsigned _sp,        \
             unsigned _bx,  unsigned _dx,  unsigned _cx,        \
             unsigned _ax,  unsigned _ip,  unsigned __cs,       \
             unsigned _flags
#elif defined __TURBOC__
#define ARGS unsigned _bp,  unsigned _di,  unsigned _si,        \
             unsigned __ds,unsigned __es,unsigned _dx,          \
             unsigned _cx,  unsigned _bx,  unsigned _ax,        \
             unsigned _ip,  unsigned __cs,unsigned _flags
#endif

/* ivstack() and appstack() macros are used for stack switching. */
#if defined _MSC_VER
#define ivstack()                                               \
    _asm {                                                      \
        _asm cli                                                \
        _asm mov WORD PTR appstack[0], sp                       \
        _asm mov WORD PTR appstack[2], ss                       \
        _asm mov sp, WORD PTR tsrstack[0]                       \
        _asm mov ss, WORD PTR tsrstack[2]                       \
        _asm sti                                                \
    }

#define appstack()                                              \
    _asm {                                                      \
        _asm cli                                                \
        _asm mov sp, WORD PTR appstack[0]                       \
        _asm mov ss, WORD PTR appstack[2]                       \
        _asm sti                                                \
    }
#elif defined __TURBOC__
#define ivstack()                                               \
    _asm {                                                      \
        cli;                                                    \
```

```
        mov WORD PTR appstack[0], sp;                   \
        mov WORD PTR appstack[2], ss;                   \
        mov sp, WORD PTR tsrstack[0];                   \
        mov ss, WORD PTR tsrstack[2];                   \
        sti;                                            \
    }

#define appstack()                                      \
    _asm {                                              \
        cli;                                            \
        mov sp, WORD PTR appstack[0];                   \
        mov ss, WORD PTR appstack[2];                   \
        sti;                                            \
    }
#endif

#if defined _DOS_NEED
extern unsigned char _far *v_indos;
extern unsigned char _far *v_incrit;

#define indos() (*(v_indos))
#define incrit() (*(v_incrit))
#endif

void _cdecl _call_intr(void (_interrupt _far *)());
#if defined __TURBOC__
    void _cdecl _chain_intr(void (_interrupt _far *)());
#endif

/* Local function prototypes. */
int   iv_sense( void );
int   iv_remove( void );
int   iv_install( void );
void iv_update( void );
int   iv_scan( int option );

/* Function prototypes */
unsigned char _far * _dos_getindos( void );
unsigned  _dos_getpsp( void );
void      _dos_setpsp( unsigned newpsp );

#endif
```

Листинг 6

```
/* Source file: VTSR.C
* Tab stops: 4
* Compilers supported:  QuickC 2.51, Microsoft C 6.0 and later,
*                       Turbo C++ 1.0, Borland C++ 2.0 and later.
* Model: Small
* Author: Dmitri Valdine.
* Email: Relcom (InterNET Department) valdine@ipmo.spb.su
* Last edition: 27-Oct-92.
* Purpose: This file should be included in your intervention
* program after SIGNATURE of the program would be defined and
* vect array would be declared and initialized.
*/

static char *IV_SIGNATURE = SIGNATURE;

/* Figure out HEAP and STACK sizes. */
#if !defined    HEAP_SIZE
#define         HEAP_SIZE   0
#endif

#if defined __TURBOC__
    #if defined STACK_SIZE
        unsigned _stklen    = STACK_SIZE;
    #endif
    #if (HEAP_SIZE == 0)
        unsigned _heaplen   = 1;
    #else
        unsigned _heaplen   = HEAP_SIZE;
    #endif
#endif

#if defined __TURBOC__ && (__TURBOC__ < 0x297)
/* Unfortunately, Turbo C++ 1.0 doesn't have _fstrcmp() function
* in it's Run-Time Library, so we have to produce version based on
* example taken from K&R bestseller.
*/
int _fstrcmp( char _far *s, char _far *t )
{
    for ( ; *s == *t; s++, t++ )
        if ( *s == '\0' )
            return 0;
    return *s - *t;
}
#endif

/* Huge pointers force compiler to do segment arithmetic for us. */
char _huge *tsrstack;
char _huge *appstack;
char _huge *tsrbottom;

/* Declare pointer to the ICB structure. */
ICB _far *picb;

/* Number of vectors to be installed. */
#define NVECTS (sizeof( vect ) / sizeof( vect[0] ))

/* This function detects whether our program was
* already installed or not. It returns:
* IV_INSTALLED   -    if copy of our program was found in memory;
* IV_ATTACHED    -    if copy of our program was found in memory and
*                     some of its vectors were grabbed by other
*                     TSR-program, loaded after ours;
* IV_NOT_FOUND -      if no copy of our program was found.
*/
int iv_sense( void )
{
    register int i;

    if ( iv_scan( IV_SENSE ) != IV_NOT_FOUND ) {
        /* Now global variable picb points to ICB structure
        * found in memory. Let's check whether our program
        * still governs all hooked interrupt vectors or not.
        * It's nesessary since we are not allowed to remove
        * our program if some other TSR-program hooked our
        * interrupts.
        */
        for ( i = 0; i < NVECTS; i++ ) {
            if ( _dos_getvect( vect[i].num ) != picb->pvect[i].new )
                return IV_ATTACHED;
        }
        return IV_INSTALLED;
    }
    return IV_NOT_FOUND;
}

/* This routine installs all interrupt vectors specified
* in the VECT array.
*/
int iv_install( void )
{
    unsigned tsrsize;
    register int i;

    /* Initialize stack and bottom of program. */
    _asm movWORD PTR tsrstack[0], sp
    _asm movWORD PTR tsrstack[2], ss
    tsrbottom = MK_FP( _psp, 0 );

#if defined _DOS_NEED
    /* Obtain adress of InDOS flag. */
    _dos_getindos();
#endif
    /* Program size is:
    *       top of stack
    *   -   bottom of program (converted to paragraphs)
    *   +   heap size (for programs compiled with Microsoft compilers)
    *   +   some extra paragraphs just to be safe.
    */
    tsrsize = ((tsrstack - tsrbottom) >> 4) + 5;

#if defined _MSC_VER
    tsrsize += (HEAP_SIZE >> 4);
#endif

    /* Our ICB will be located with the ICB_OFFSET
    * from the beginning of PSP of our program.
    */
    picb = MK_FP( _psp, ICB_OFFSET );

    /* Fill in the ICB structure. */
    picb->signature     = IV_SIGNATURE;
    picb->pvect         = vect;
#if defined _UPDATE_NEED
    picb->ivdata        = &ivdata;
#endif

    for ( i = 0; i < NVECTS; i++ ) {

        /* Obtain the previos contents of interrupt vector. */
        *(vect[i].old) = _dos_getvect( vect[i].num );

        /* Install our own interrupt handler. */
        _dos_setvect( vect[i].num, vect[i].new );
    }

    /* Free environment if nesessary. */
#if defined _FREE_ENVIRONMENT
    _dos_freemem( peek( _psp, 0x2c ));
#endif

    /* Terminate and stay resident. */
    _dos_keep( OK, tsrsize );

    /* Shouldn't come here. */
    return ERROR;
}

#if defined _UPDATE_NEED
/* This routine updates fields in IVDATA structure
* of already installed copy of our program.
*/
void iv_update( void )
{
    *((IVDATA _far *) picb->ivdata) = ivdata;
}
#endif

/* This routine restores previous state of the interrupt vectors
* and releases the memory occupied by the program.
*/
int iv_remove( void )
{
    register int i;

    /* First, restore the previous contents of the interrupt vectors. */
    for ( i = 0; i < NVECTS; i++ )
        _dos_setvect( picb->pvect[i].num, *(picb->pvect[i].old ));

    /* Now release the memory occupied by our program. */
    return iv_scan( IV_REMOVE );
}

/* This routine scans MCB chain and depending of option value
* either looks for ICB adress or
```

```
 * releases the memory occupied by the program.
 */
int iv_scan( int option )
{
    unsigned mcbseg;
    MCB _far *pmcb;

    /* Obtain adress of the first memory block in the DOS chain.
     * We're using undocumented DOS function here.
     */
    _asm {
        push es
        mov ah, 52h
        int  21h
        mov ax, es:[bx-2]
        mov mcbseg, ax
        pop  es
    }

    /* Now loop through the chain of DOS memory blocks. */
    for ( EVER ) {
        /* Obtain MCB contents. */
        pmcb = MK_FP( mcbseg, 0 );

        /* If it is not a legitimate DOS memory block,
         * then break with an ERROR.
         */
        if ( pmcb->ident != 0x4d && pmcb->ident != 0x5a )
            return ERROR;

        if ( option == IV_REMOVE ) {
            /* If this block belongs to our intervention program,
             * then free the block.
             */
            if ( pmcb->owner_psp == FP_SEG( picb )
                    && _dos_freemem( mcbseg + 1 ) != 0 )
                return ERROR;
        }
        else {
            /* If this block is a PSP block,
             * then check if it contains our particular ICB.
             */
            if ( pmcb->owner_psp == mcbseg + 1 ) {
                picb = MK_FP( pmcb->owner_psp, ICB_OFFSET );

                if ( _fstrcmp( picb->signature, IV_SIGNATURE ) == 0 )
                    return IV_INSTALLED;
            }
        }

        /* If it is not the last block in the chain, then break. */
        if ( pmcb->ident == 0x4d )
            mcbseg += pmcb->size + 1;
        else
            break;
    }

    return ( option == IV_REMOVE ) ? OK : IV_NOT_FOUND;
}

#if defined _DOS_NEED
unsigned char _far *v_indos  = (void _far *) 0;
unsigned char _far *v_incrit = (void _far *) 0;

unsigned char _far *_dos_getindos()
{
    /* Preserve ES and SI registers. */
    _asm    push es
    _asm    push si

    /* DOS function 0x34: return address of InDOS flag. */
    _asm {
        mov ah, 34h
        int  21h
        mov word ptr v_indos, bx
        mov word ptr v_indos+2, es
    }

    /* Undocumented DOS function 0x5d06:
     * return address of DOS Swappable Area.
     * Note: v_indos - 1 must be equal to v_incrit !
     * Since the call is undocumented there is a few chances
     * that it would fail, so you should choose what to do in that case.
     */
    _asm {
        push ds
        mov ax, 5d06h
        int  21h
        push ds
        pop  es
        pop  ds
        mov word ptr v_incrit, si
        mov word ptr v_incrit+2, es
    }

    /* Restore ES and SI registers. */
    _asm    pop si
    _asm    pop es

    return v_indos;
}

unsigned _dos_getpsp( void )
{
    static union REGS regs;

    /* DOS function 0x62 (Return PSP) is available (and documented)
     * for DOS versions 3.00 and later. Function 0x51 is available,
     * but not documented, for all versions of DOS, so prefer
     * function 0x62 when it is available.
     */
    regs.h.ah = (unsigned char) (( _osmajor < 3 ) ? 0x51 : 0x62 );
    int86( 0x21, &regs, &regs );

    return regs.x.bx;
}

void _dos_setpsp( unsigned newpsp )
{
    /* DOS function 0x50: Set current PSP (an undocumented function). */
    _asm {
        mov ah, 50h
        mov bx, newpsp
        int  21h
    }
}
#endif

#if defined _MSC_VER
/* Supress the null-pointer check, environment-processing
 * and float pointer trapping routines. It will slightly
 * reduce the program size.
 */
void _nullcheck( void ) { }
void _setenvp( void ) { }
void _fptrap( void ) { }
#endif
```

Листинг 7

```
/* Header file: VMISC.H
 * Tab stops: 4
 * Compilers supported:  QuickC 2.51, Microsoft C 6.0 and later,
 *                       Turbo C++ 1.0, Borland C++ 2.0 and later.
 * Author: Dmitri Vaidine.
 * Email: Relcom (InterNET Department) vaidine@ipmo.spb.su
 * Last edition: 27-Oct-92.
 * Purpose: This file defines some useful values and macros for C programs.
 */

#ifndef DEF_VMISC
#define DEF_VMISC

#define NO          0
#define YES         1

#define FALSE       0
#define TRUE        1

#define ERROR       (-1)
#define OK          0

#define EVER        ;;

#ifndef MK_FP
#define MK_FP(seg,off) ((void _far *)                                   \
    ((((unsigned long)(unsigned int)(seg)) << 16L) |                    \
    ((unsigned long)(unsigned int)(off))))
#endif

#ifndef peekb
#define peekb(seg,off) (*((unsigned char _far *) MK_FP((seg),(off))))
#endif

#ifndef peek
#define peek(seg,off) (*((unsigned int _far *) MK_FP((seg),(off))))
#endif

#ifndef pokeb
#define pokeb(seg,off,c)(*((unsigned char _far *)                       \
    MK_FP((seg),(off))) = (unsigned char)(c))
#endif

#ifndef poke
#define poke(seg,off,c)(*((unsigned _far * )                            \
    MK_FP((seg),(off))) = (unsigned)(c))
#endif

#define lowbyte(a)  ((a) & 0x00ff)
#define highbyte(a)(((a) >> 8) & 0x00ff)

/* Parsing macros. */
#define StartParsing                                                    \
    while ( --ac > 0 && strchr( "-/", **++av )) {                       \
        char *p = *av;                                                  \
        while ( *++p ) {                                                \
            switch( *p ) {

#define FinishParsing                                                   \
            default:                                                    \
                cputs( "Error: unknown switch " );                      \
                putch( *p );                                            \
                cputs( ".\n\r" );                                       \
                goto usage;                                             \
            }                                                           \
        }                                                               \
    }

/* Make some substitutions for Borland compilers. */
#if defined __TURBOC__
    #define _cdecl      cdecl
    #define _far        far
    #define _near       near
    #define _huge       huge
    #define _asm        asm
#endif

/* Define BEEP() macro. */
#if defined _MSC_VER
#define BEEP()      _asm {                                              \
                        _asm sub bx, bx         /* Page 0       */      \
```

```
                    _asm movax, 0E07h       /* TTY bell    */      \
                    _asm int  10h           /* BIOS 10     */      \
                }
#elif defined __TURBOC__
#define BEEP()      _asm {                                        \
                    sub  bx, bx;            /* Page 0      */     \
                    mov ax, 0E07h;          /* TTY bell    */     \
                    int  10h;               /* BIOS 10     */     \
                }
#endif
#endif
```

Листинг 8

```
/* Source file: ALARM.C
* Tab stops: 4
* Compilers supported:  QuickC 2.51, Microsoft C 6.0 and later,
*                       Turbo C++ 1.0, Borland C++ 2.0 and later.
* Model: Small
* Author: Dmitri Valdine.
* Email: Relcom (InterNET Department) valdine@ipmo.spb.su
* Last edition: 27-Oct-92.
* Purpose: This program shows technique of creating TSR-programs
* in C language and is based on a simple example taken from QuickC 2.51
* on-line help.
*/

/* Allow our Alarm program to be updated if it was already installed. */
#define _UPDATE_NEED

#include "vtsr.h"

/* Signature of the program. */
#define SIGNATURE "Alarm 1.0"

/* Define stack size. Heap size is equal to zero by default. */
#define STACK_SIZE    256

#define TICKPERMIN   1092L
#define MINPERHOUR 60L

/* Variables for storing adresses of old interrupt handlers. */
void (_interrupt _far *oldtimer)( void );
void (_interrupt _far *oldvideo)( void );

/* Prototypes for interrupt functions */
void _interrupt _far newtimer( void );
void _interrupt _far newvideo( void );

/* Declare and initialize vect array. */
IVECT vect[] = {
      { 0x08, &oldtimer, newtimer },
      { 0x10, &oldvideo, newvideo },
};

/* Define structure which will contain some specific data,
* nessessary for our ALARM program (goaltick to ring alarm
* and internal flag indicating that alarm was already rang).
* We should do it since we want to allow our intervention
* program to be updated.
*/
typedef struct {
      long goaltick;
      int  ftimesup;
} IVDATA;

IVDATA ivdata;

/* Variables that will be accessed inside TSR must be global. */
long _far *pcurtick = (long _far *)0x0000046cL;

/* Busy flag. */
int video_busy = NO;

#include "vtsr.c"

main( int ac, char **av )
{
      int remove_it = NO;

      cputs( "Alarm 1.0   Oct.92\n\rWritten by Dmitri Valdine\n\r" );

      /* Parse command-line arguments. */
      StartParsing
            case 'r':
            case 'R':
                  remove_it = YES;
                  break;
      FinishParsing

      /* If remove flag wasn't set then... */
      if ( !remove_it ) {
            long minute, hour;

            /* Check command-line syntax. */
            if ( ac != 1 ) {
usage:
                  cputs( "usage: alarm [-r] [<hh:mm>]\n\r" );
                  cputs( "where <hh:mm> is time to ring alarm.\n\r" );
                  return ERROR;
            }

            /* Check the time syntax. */
            {
            char *p = *av;

            if ( p[2] != ':' || p[5] != '\0'
                        || (p[2] = '\0', strcmp( p, "00" )) < 0
                        || strcmp( p,  "23" ) > 0
                        || strcmp( p+3, "00" ) < 0
                        || strcmp( p+3, "59" ) > 0 ) {
                  cputs( "Bad time syntax.\n\r" );
                  return ERROR;
            }
            hour   = atol( p );
            minute = atol( p+3 );
            }

            /* Convert time to ticks past midnight.
            * Time must be later than current time.
            * Fill in the ivdata structure.
            */
            ivdata.goaltick = (hour * MINPERHOUR * TICKPERMIN)
                              + (minute * TICKPERMIN);
            if( *pcurtick > ivdata.goaltick ) {
                  cputs( "It's past that time now.\n\r" );
                  return ERROR;
            }
            ivdata.ftimesup = NO;
      }
      /* Give all routine work to install() macro. */
      install( Alarm );

      return OK;
}

/* Our timer interrupt compares current time to goal. If earlier,
* it just continues. If later, it beeps and sets a flag to quit checking.
*/
void _interrupt _far newtimer()
{
      if( ivdata.ftimesup )
            _chain_intr( oldtimer );
      else {
            /* First, execute the original timer interrupt. */
            _call_intr( oldtimer );

            /* Activate if two conditions are met: First, it's past time for
            * the alarm. Second, we are not in a video interrupt. Checking
            * the video interrupt prevents the rare but potentially
            * dangerous case of calling INT 10 to beep while INT 10 is
            * already running.
            */
            if( (*pcurtick > ivdata.goaltick) && !video_busy ) {
                  /* Set flag so we'll never return.
                  * This will also prevent us from reentering this code
                  * while the beeping is in progress.
                  */
                  ivdata.ftimesup = YES;

                  /* Reenable interrupts
                  * so ticks occured while beeping won't be lost.
                  */
                  _asm sti

                  /* Actually beep 3 times. */
                  BEEP();
                  BEEP();
                  BEEP();
            }
      }
}

/* The sole purpose of this handler is to maintain
* busy flag, while video interrupt is in progress.
*/
void _interrupt _far newvideo( void )
{
      /* Set the inside flag. */
      video_busy++;

      /* Call the original interrupt. */
      _call_intr( oldvideo );

      /* Clear the inside flag. */
      video_busy--;
}
```

Листинг 9. Файл BC2_0.BAT

```
@echo off
tasm -ml -dBC callintr.asm
bcc -O alarm.c getch.c callintr.obj
```

Листинг 10. Файл MSC6_0.BAT

```
@echo off
cl /c /AS /W2 /Ze /Os alarm.c
masm /Ml /DMSC callintr.asm;
link /NOE /ST:0x100 alarm+callintr;
```

Листинг 11. Файл QC2_51.BAT

```
@echo off
qcl /c /AS /W2 /Ze /Ox alarm.c
qcl /c /DMSC /AS /W2 callintr.asm
qlink /NOE /ST:0x100 alarm+callintr;
```

Листинг 12. Файл TCPP1_0.BAT

```
@echo off
tasm -ml -dBC callintr.asm
tcc -lt alarm.c getch.c callintr.obj
```

Фирма ИНСАЙТ INSIGHT Comp.

Дистрибьютер **CLARION SOFTWARE** Corp.

CLARION · TopSpeed

Версии 2.1 и 3.0 Pascal, Modula-2, C, C++

Идеальные средства создания заказного программного обеспечения.

Семинары CLARION User Group;
для официальных пользователей -
"горячая линия"; учебные курсы.

Предлагаем программное обеспечение фирм
Borland, Symantec, Microsoft, Corel, WordPerfect

103489, Москва, Зеленоград, корп. 602а
тел. 536-69-00 факс 534-93-46
E-Mail: puh@insight.pccentre.msk.su | 2:5020/11
CompuServe: 100140, 1224

МАГАЗИН
+LINK

ЭТО ВСЁ ДЛЯ КОМПЬЮТЕРНЫХ СЕТЕЙ И КОМПЬЮТЕРОВ

Тел. 263-94-40 *Факс 267-58-80*
Ново-Басманная ул., 20

Вы можете приобрести в нашем магазине любое оборудование к Вашему компьютеру:

- Сетевые адаптеры ArcNet, EtherNet.
- Модемы и факс-модемы от 2400 до 14400.
- Картриджи для матричных принтеров.
- Системы бесперебойного питания UPS APC.
- Комплектующие к ПЭВМ.
- Компьютеры.
- Стриммеры.
- Сканеры и прочее.

Мы ждем Вас!

Лучшие цены за лучшее оборудование

Для эффективной работы необходимо использование качественного и надежного оборудования. Компания Демос совместно с австрийскими фирмами **APS-COM** и **FORTUNA** предлагает:

- компьютеры от 386SX до Pentium
- лазерные принтеры Hewlett-Packard и Canon
- компьютеры-блокноты
- компьютеры класса booksize
- стримеры емкостью от 125Мбайт до 8Гбайт
- цветные и черно-белые сканеры
- графопостроители
- самые дешевые сетевые адаптеры Ethernet, проверенные с Novell NetWare, TCP/IP, DEC Net, MS Windows for Workgroup, NetBIOS
- любое оборудование и инструменты для создания локальных сетей
- настольные издательские системы
- автоматические переключатели для использования одного принтера несколькими компьютерами
- разнообразные принадлежности для компьютеров — звоните

Canon

HEWLETT PACKARD Authorized Dealer

Сегодня мы поставляем только лучшее оборудование, делая все для снижения его цены.

demos™
APS-COM

Телефоны: (095) 231-21-29, 231-63-95, 233-05-92, 233-02-42, 231-60-02
Факс: (095) 233-50-16
E-mail: info@hq.demos.su

Знак и логотип **demos** — торговая марка фирмы Демос

Звуковые устройства

Д. Солдатенков

А вы ноктюрн сыграть смогли бы
На флейтах водосточных труб?

В. Маяковский

В области графики за последние годы наметился явный прогресс. Общепризнанным минимумом стал адаптер VGA. Новые отечественные программы перестают поддерживать адаптеры типа CGA и EGA. А вот звук все еще находится в первобытном состоянии. Может быть, обзор наиболее распространенных звуковых устройств и музыкальных форматов поможет кому-то дополнить встроенный динамик полноценным звуковым устройством и обрести еще один мир, не менее удивительный, чем мир компьютерной графики.

Внутренний динамик (PC Speaker)

Самым главным и, пожалуй, единственным достоинством вышеупомянутого звукового устройства является то, что им оснащен каждый компьютер и уж если программа его поддерживает, то худо-бедно, но она сможет вести звуковое сопровождение на любом компьютере. Это особенно актуально для нашей родины, где специализированные звуковые платы — большая редкость.

Сложности в создании звука связаны с тем, что на динамик можно посылать сигнал только двух уровней — 0 или 1. Но даже если смириться с тем, что не удастся поставить мембрану динамика в промежуточное положение, существует несколько способов создания звука на внутреннем динамике, которые родились на заре звуковой технологии. Все они используют сигнал прямоугольной формы. Такие сигналы легко узнать на слух по характерному тембру.

Вся ранняя компьютерная музыка состояла из суммы прямоугольных сигналов. Были созданы довольно удачные программы для ее написания (редактор СП «Диалог»). Но те гудки, из которых состоит данная музыка, не могут соперничать со звуком реальных инструментов, и музыкальные технологии, пришедшие с компьютеров Amiga, не оставили никаких шансов прежним гуделкам. Эти технологии предполагают использование так называемых оцифровок.

Под словом «оцифровка» понимается амплитуда звукового сигнала, измеренная через одинаковые промежутки времени и преобразованная при помощи АЦП в целое число. Частоту, с которой производится измерение амплитуды, будем называть частотой выборки.

К счастью, у внутреннего динамика предусмотрена возможность воспроизведения оцифровок при помощи метода широтно-импульсной модуляции (ШИМ). Суть метода заключается в том, что можно управлять положением мембраны динамика за счет изменения ширины импульсов. Предположим, что мы подаем на динамик прямоугольные импульсы с постоянной частотой и переменной скважностью, причем частота импульсов превосходит максимальную частоту, воспроизводимую динамиком. В этом случае динамик будет усреднять протекающий через него ток и высота положения мембраны будет пропорциональна среднему току через динамик, то есть пропорциональна ширине импульсов.

Данные, посылаемые в этом случае на динамик, ничем не отличаются от данных, полученных в результате аналого-цифрового преобразования, и могут быть использованы без изменения для вывода на Sound Blaster или Covox. Этот приятный факт позволяет создавать эффективные программы, поддерживающие сразу несколько звуковых устройств.

Для получения удовлетворительного качества звука на внутреннем динамике частота выборки (sampling rate) должна быть больше 16 кГц. Это связано как раз с тем, что мы можем управлять

не положением мембраны динамика, но лишь шириной импульсов, через него протекающих. Если снизить частоту выборки, то тут же возникнет неприятный свист, громкость которого будет возрастать с дальнейшим снижением частоты.

Этот способ создания звука используется многими программами, воспроизводящими звуковые эффекты на внутреннем динамике. Применяется он также и программами, поддерживающими внутренний динамик и проигрывающими музыку, написанную в формате MOD и STM.

Для того чтобы получить оцифровку, не всегда требуется АЦП. В некоторых случаях ее можно просто рассчитать, как это делается в приводимом листинге. Программа имитирует бой часов. Вывод идет на внутренний динамик, при этом на экране высвечивается осциллограмма сигнала. Приводимый пример является фрагментом пакета «Маэстро», разработанного фирмой Maddox Games и распространяемого журналом «Монитор». Пакет поддерживает внутренний динамик, Covox и Sound Blaster.

Covox

Итак, на смену гудкам пришли оцифровки. Сначала они использовались только для звуковых эффектов, но вскоре для компьютеров Amiga был создан звуковой процессор, который позволил реализовать музыку из оцифровок.

Если записать оцифровку какого-либо инструмента на фиксированной ноте, а потом начать ее растягивать и сжимать, то можно получить все требуемые ноты. Основная трудность заключается в том, что масштабировать оцифровку надо во время воспроизведения мелодии. Потом сигналы, полученные от различных инструментов, просто суммируются.

Именно этим и занимается звуковой сопроцессор на Amiga. Для музыки, создаваемой из оцифровок, родился свой формат — MOD, а также множество редакторов и воспроизводящих программ.

На IBM PC, как известно, звукового сопроцессора нет, но играть красивую музыку очень хочется, поэтому пришлось нагрузить единственный имеющийся в распоряжении процессор. Минимальное быстродействие, которое требуется для воспроизведения музыки из оцифровок, причем самого низкого качества, — 8 МГц. При этом о каком-либо параллельном процессе лучше даже и не думать.

Тем не менее с ростом производительности машин музыка из оцифровок распространилась и на IBM PC. Возникли новые форматы: STM, 669, OCT, ... Многие компьютерные игры воспроизводят такую музыку в фоновом режиме.

Но для того чтобы слушать музыку из оцифровок, внутреннего динамика уже недостаточно. Нужна нормальная акустическая система и устройство, которое позволило бы вывести на нее звуковой сигнал, например, Covox.

Под этим названием подразумевается целое семейство различных звуковых устройств, использующих параллельный порт для подключения ЦАП. Covox позволяет воспроизводить оцифровки через качественную акустическую систему и отличается низкой стоимостью (рис. 1). Частота выборки, в отличие от внутреннего динамика, может быть снижена. Скорость работы значительно превышает скорость платы Sound Blaster в непосредственном режиме (об этой плате речь пойдет далее).

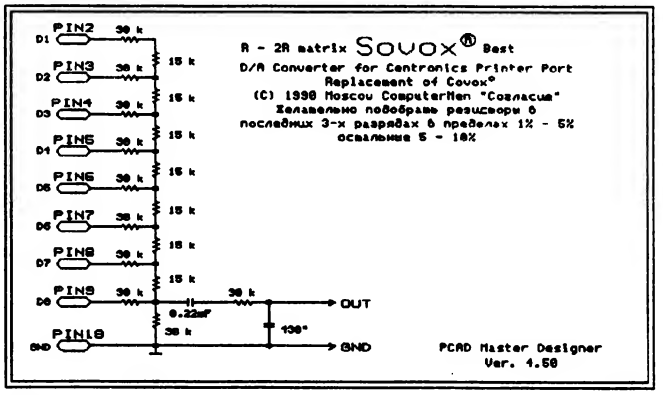


Рис. 1

В связи с тем, что Covox очень просто изготовить самостоятельно, это устройство быстро распространилось у нас в стране. (Цена ≈ 5$).

AdLib

Но путь оцифровок — не единственный.

Широкое распространение получила плата AdLib, вернее, микросхема FM-синтезатора фирмы YAMAHA, которая встраивается последнее время во все звуковые платы для того, чтобы обеспечить их совместимость с платой AdLib.

Эта микросхема содержит 18 цепей, способных генерировать сигнал вида A sin (Wt). Управляя микросхемой программно, цепи можно соединять различным образом, менять коэффициенты у амплитуды и частоты. При этом синтезируемый звук может напоминать звучание различных музыкальных инструментов и даже голоса человека. Микросхема имеет 3 режима, различающихся между собой способом использования цепей. Два режима предназначены для воспроизведения музыки и один для синтеза речи.

Огромное распространение этой микросхемы связано прежде всего с тем, что она практически не отбирает время у процессора во время воспроизведения музыки и звуковое сопровождение становится возможным даже в играх, требующих больших расчетов.

Сразу следует сказать, что музыка FM-синтезатора по разнообразию и окраске инструментов существенно уступает музыке, использующей оцифровки реальных инструментов, такие как MOD и STM. Чистота же звучания у FM-музыки очень высока. Аналог возможен для музыки реальных инструментов только при частоте выборки около 40 кГц. Теорема Котельникова неумолима.

Существует несколько стандартных музыкальных форматов для записи FM-музыки. Самым распространенным из них является MIDI (Music International Digital Interface), который принят как стандарт для Microsoft Windows. Музыка, записанная в этом формате, состоит из так называемых MIDI событий. Эти события являются общепринятыми для всех электронных музыкальных инструментов, поддерживающих MIDI стандарт, и, вообще говоря, с компьютером не связаны. Они описывают всевозможные действия

музыканта и только действия музыканта, а не то, как звучит инструмент.

Звуковые платы, имеющие MIDI вход, можно соединить с электронным музыкальным инструментом, имеющим фортепианную клавиатуру, и записать музыку с него в MIDI формате.

Недостатком формата MIDI является то, что он не содержит в себе никакой информации об инструментах. Каждая система, воспроизводящая MIDI музыку, устанавливает набор инструментов на свое собственное усмотрение. Поэтому одна и та же мелодия звучит совершенно по-разному в разных системах.

От этого недостатка избавлены два других формата FM-музыки — CMF и ROL. Они содержат в себе полную информацию об инструментах и воспроизводятся одинаково в любой ситуации, но CMF жестко ориентирован на использование Sound Blaster, ROL — на использование AdLib.

Очевидно, что плата AdLib не рассчитана на воспроизведение оцифровок, но люди любят оцифровки и поэтому разработали способ их воспроизведения через AdLib. Он чем-то напоминает воспроизведение через встроенный динамик (PC speaker). (Цена ≈ 50$).

Sound Blaster

Это самое распространенное семейство звуковых плат. Последние версии могут воспроизводить музыку и звуковые эффекты в стереорежиме. Плата содержит FM-синтезатор, цифровой звуковой процессор и миксер.

FM-синтезатор позволяет эмулировать плату AdLib и воспроизводить FM-музыку как в моно-, так и в стереорежиме. Количество колебательных цепей сильно увеличено в последних версиях, что позволяет на 1 музыкальный инструмент использовать до 4 цепей (или операторов). Такая музыка называется 4-операторной. Звучание инструментов гораздо ярче, чем в ранее используемой 2-операторной музыке.

Кроме того, плата имеет MIDI вход, который позволяет вводить музыку с фортепианной клавиатуры синтезатора, и MIDI выход, который позволяет воспроизводить музыку через внешний синтезатор.

Звуковой процессор (Digital Signal Processor — DSP) позволяет воспроизводить и записывать оцифровки. Для этого у него есть два режима — непосредственный и прямого доступа к памяти. В непосредственном режиме программа должна так же, как и при работе с Covox, посылать через равные промежутки времени данные в надлежащий порт. При этом перед посылкой каждого следующего байта данных необходимо удостовериться в готовности платы.

В режиме прямого доступа к памяти (ПДП, или DMA) плата без участия процессора берет данные из указанной области и воспроизводит с определенной частотой выборки. Этот режим гораздо экономичнее первого и используется чаще. Именно в этом режиме работает пакет «Маэстро», воспроизводя музыку в STM-формате и звуковые эффекты через Blaster.

При помощи Sound Blaster можно записывать оцифровки. Плата имеет линейный и микрофонный вход.

DSP может аппаратно «разжимать» звуковые данные. Он делает это прямо во время воспроизведения и поддерживает механизмы сжатия 3 типов: 2-1, 3-1, 4-1.

Последние версии платы могут воспроизводить и записывать стереозвуки. Микросхема миксера управляет режимом, фильтрами низких и высоких частот, громкостью. На плате находятся переключатели, позволяющие изменять адреса портов, через которые происходит управление платой, номер аппаратного прерывания и канал прямого доступа к памяти. Имеется также усилитель — по 4 Ватта на канал и порт джойстика.

Плата SB 16 использует для задания амплитуды сигнала не 8, а 16 битов, в связи с этим плата задействует 2-ю микросхему DMA, что всегда сопряжено с определенным риском. Вторая микросхема часто не работает. Об этом предупреждает даже руководство по эксплуатации SB 16.

По количеству программного обеспечения Sound Blaster оставит далеко позади все остальные звуковые карты. Трудно себе представить, что кто-нибудь рискнет выпустить на рынок звучащую программу, несовместимую с Sound Blaster. Такая программа обречена на неудачу. (Цена ≈ 150-400$).

Gravis UltraSound

Казалось бы, все есть у Sound Blaster. Но эта плата так и не смогла обеспечить обладателю самой современной IBM PC возможности обладателя самой первобытной Amiga. Людям нравится музыка оцифровок. Воспроизведение музыки MOD формата через Sound Blaster даже на машине 40 МГц сделает невозможной быструю графику.

Плата Gravis UltraSound (GUS) несет на борту до 1 М памяти, куда программы записывают оцифровки, и делает, по сути, то, что и звуковой сопроцессор Amiga. Эта плата стоит меньше Sound Blaster и очень бурно распространяется.

GUS может эмулировать Sound Blaster при помощи резидентного драйвера. Существуют драйверы под Windows. Музыку MIDI формата программы этой платы исполняют при помощи оцифровок, а проигрывание форматов MOD и STM совсем не занимает времени основного процессора.

GUS использует 16-битовые оцифровки, но может работать и с 8-битовыми. Сейчас GUS интенсивно развивается. (Цена ≈ 156-220$).

Если Вам захотелось включить музыку реальных инструментов в свои программы, то Вы можете приобрести пакет «Маэстро» в журнале «Монитор». Он воспроизводит звуковые эффекты через 3 устройства параллельно с музыкой и без нее. Если же у вас появятся более серьезные проблемы со звуком, то Вы можете связаться с фирмой Maddox Games по телефонам (095) 193-71-74, 562-63-02 или по электронной почте oleg@maddox.msk.su. ■

Редакция журналов «Монитор» и «Монитор-Аспект» приглашает к сотрудничеству специалистов, желающих принять участие в работах по тестированию программных продуктов и аппаратуры.

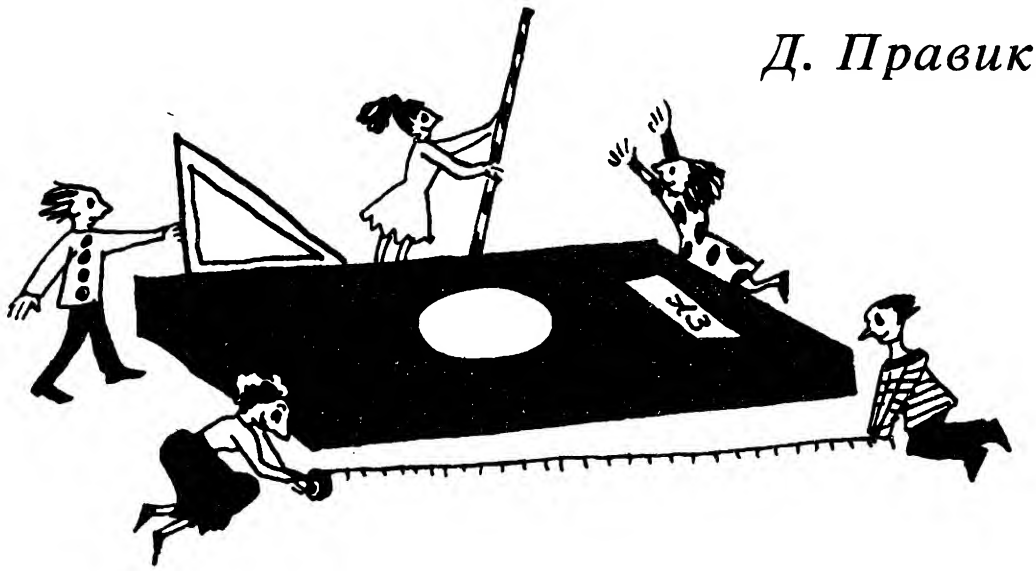
Обращаться по тел. 335-68-31.
Дмитрий Бройтман

Листинг

```
program Clock;

uses Crt;

const
    Divisor: Word = $4B;
    BuffSize = 20000;

type
    PSoundData = ^TSoundData;
    TSoundData = array [0..BuffSize-1] of Byte;
    TScr = array [0..25, 0..79] of record Ch, Atr: Byte end;

var
    BuffPtr: Pointer;
    SoundData: PSoundData;
    OldInt8: Pointer;
    Scr: TScr absolute $B800:0;
    Bum: Byte;

const
    { Переводит значение амплитуды в сигнал, посылаемый на динамик }
    SpeakerTable: array [1..256] of Byte = (
    $01,$01,$01,$01,$01,$01,$02,$02,$02,$02,$02,$02,$02,$02,$02,$02,
    $02,$02,$02,$03,$03,$03,$03,$03,$03,$03,$03,$03,$03,$04,$04,
    $04,$04,$04,$04,$04,$04,$04,$04,$04,$04,$05,$05,$05,$05,$05,$05,
    $05,$05,$05,$05,$05,$06,$06,$06,$06,$06,$06,$06,$06,$06,$06,$06,
    $06,$06,$06,$07,$07,$07,$07,$07,$07,$07,$07,$07,$07,$08,$08,
    $08,$08,$08,$08,$08,$08,$08,$08,$08,$08,$09,$09,$09,$09,$09,
    $09,$09,$0A,$0A,$0A,$0A,$0B,$0B,$0C,$0C,$0D,$0E,$0E,$0F,$10,$11,
    $11,$12,$13,$14,$15,$16,$17,$18,$1A,$1B,$1C,$1D,$1E,$1F,$21,$22,
    $23,$24,$26,$27,$28,$29,$2B,$2C,$2D,$2E,$30,$31,$32,$33,$34,$35,
    $36,$37,$39,$3A,$3A,$3B,$3C,$3D,$3E,$3F,$40,$40,$41,$42,$42,$43,
    $43,$43,$43,$43,$43,$44,$44,$44,$44,$44,$44,$44,$44,$44,$44,$44,
    $44,$44,$45,$45,$45,$45,$45,$45,$45,$45,$45,$45,$45,$46,$46,$46,
    $46,$46,$46,$46,$46,$46,$46,$46,$46,$46,$47,$47,$47,$47,$47,$47,
    $47,$47,$47,$47,$47,$48,$48,$48,$48,$48,$48,$48,$48,$48,$48,$48,
    $48,$48,$49,$49,$49,$49,$49,$49,$49,$49,$49,$49,$49,$4A,$4A,$4A,
    $4A,$4A,$4A,$4A,$4A,$4A,$4A,$4A,$4A,$4A,$4A,$4B,$4B,$4B,$4B,$4B,$4B
    );

procedure SpeakerStop; assembler;
    { Возвращает таймер в исходное состояние }
asm
    push   ds
    push   es
    xor    ax, ax
    mov    es, ax
    cli
    lds    ax, OldInt8
    mov    es:[8*4], ax
    mov    es:[8*4+2], ds
    mov    al, 36h
    out    43h, al
    mov    al, 0
    out    40h, al
    out    40h, al
    in     al, 61h
    and    al, 0FCh
    out    61h, al
    sti
    pop    es
    pop    ds
end;

procedure SpeakerHandler; assembler;   { Обработчик прерывания }
asm
    push   ax
    push   bx
    push   dl
    push   es
    push   ds
    mov    ax, seg @Data
    mov    ds, ax
    les    dl, SoundData
    mov    al, byte ptr es:[dl]
    mov    bx, offset SpeakerTable
    xlat
    out    42h, al
    inc    dl
    cmp    dl, BuffSize
    jl     @@Done
    mov    dl, 0
    inc    Bum
    @@Done:
    mov    word ptr SoundData, dl
    mov    al, 20h
    out    20h, al
    pop    ds
    pop    es
    pop    dl
    pop    bx
    pop    ax
    iret
end;

procedure SpeakerRun; assembler;
                              { Приводит микросхему таймера в надлежащее}
asm                           { состояние и перехватывает 8 прерывание }
    mov    al, 0B0h
    out    43h, al
    mov    al, 1
    out    42h, al
    mov    al, 0
    out    42h, al
    in     al, 61h
    or     al, 3
    out    61h, al
    mov    al, 90h
    out    43h, al
    xor    ax, ax
    mov    es, ax
    mov    ax, es:[8*4]
    mov    bx, es:[8*4][2]
    mov    word ptr OldInt8, ax
    mov    word ptr OldInt8[2], bx
    mov    ax, offset SpeakerHandler
    cli
    mov    es:[8*4], ax
    mov    es:[8*4][2], cs
    mov    al, 36h
    out    43h, al
    mov    ax, Divisor
    out    40h, al
    mov    al, ah
    out    40h, al
    sti
end;

procedure PrepData;       { Готовит оцифровку }
var
    I: Word;
    RelCount: Real;
    SCount: Word;
begin
    WriteLn ('Готовится оцифровка.');
    SoundData := Ptr (Seg (BuffPtr^), 0);
    for I := 0 to BuffSize-1 do begin
            RelCount := I/BuffSize;
            SoundData^[I] := Round ($80 + 127/(1+RelCount)*Sin (I/15));
            SCount := 1+Trunc (RelCount*79);
            GoToXY (SCount, 3); Write ('');
    end;
end;

procedure Show;           { Осциллограф }
var
    I, J: Byte;
    Dat: Word;
    MSP, LSP: Byte;
    SD: PSoundData;
begin
    SD := SoundData;
    for I := 0 to 79 do begin
            Dat := (SD^[I] * 40) shr 8;
            MSP := Dat shr 1; LSP := Dat and 1;
            for J := 0 to 20 do
                with Scr [5 + J, I] do begin
                    if J = MSP then
                        case LSP of
                            1: Ch := 220;
                            0: Ch := 223;
                        end
                    else Ch := 32;
                    Atr := Bum + 1;
                end;
    end;
end;

begin
    ClrScr;
    GetMem (BuffPtr, BuffSize+16);
    PrepData;
    Bum := 0;
    SpeakerRun;
    repeat
        Show;
    until Bum = 12; { Часы бьют 12 раз }
    SpeakerStop;
    FreeMem (BuffPtr, BuffSize+16);
end.
```

Измерение длины дорожки дискеты

Д. Правиков



Одним из подходов, применяемых при создании некопируемых ключевых дискет, является контроль длины дорожки. Здесь под длиной понимается совокупный объем служебной информации и данных, задаваемый при операции форматирования. Соответственно в качестве единиц измерения используются бит и байт. Длина дорожки напрямую зависит от скорости вращения дисковода, на котором она была отформатирована. Из-за разброса параметров различных дисководов (официально допустимое расхождение составляет 1%) длина дорожки считается уникальной характеристикой [1]. Более того, в [1] рекомендуется специально изменять скорость вращения с целью сделать невозможным подбор дисковода с подобным характеристиками.

Измерение длины может проводиться либо с помощью операции чтения всей дорожки, либо через чтение последнего сектора, записанного с кодом длины, большим, чем код длины форматирования. Например, для дискет объемом 360 Кбайтов можно отформатировать дорожку на 10 секторов и у последнего сектора указать код длины дорожки N=4. Такой прием позволяет получить доступ к служебным полям в районе индексного отверстия и, следовательно, определить длину дорожки. Причем в [2] отмечается, что длина может контролироваться «с точностью до одного байта. Более высокая точность недостижима из-за разброса параметров контроллеров НГМД». К сожалению, авторы не привели обоснования своей точки зрения, которая представляется более чем спорной.

Для того чтобы понять, как можно измерить длину дорожки точнее, давайте рассмотрим следующую модель операции форматирования. После получения сигнала о прохождении индексного отверстия контроллер начинает записывать на дорожку служебную информацию и продолжает запись до тех пор, пока не будет получен повторный сигнал о прохождении индексного отверстия. В этот момент запись на дискету прерывается. Служебное поле, записью которого начинается дорожка, называется GAP4A, а заканчивается дорожка служебным полем GAP4B. На рис. 1 представлено расположение этих полей в области индексного отверстия.

Рис. 1

Оба поля, как GAP4A так и GAP4B, заполняются символом N (4Eh) и при рассмотрении их в качестве содержимого большого сектора должны сливаться. На практике, как правило, наблюдается другая картина. Например, переход GAP4A в GAP4B может быть виден как следующая последовательность байтов в шестнадцатеричном представлении:

4E4E4E49C9C9...

Эта последовательность трактуется следующим образом: 4E принадлежит GAP4B, а все остальное — GAP4A, в котором при форматировании было записано 4E, но это значение искажается прохождением через индексное отверстие. Зная местоположение начала области данных последнего сектора (оно легко вычисляется через параметры форматирования) и определив местоположение последнего 4E внутри сектора, мы получаем длину дорожки с точностью до байта. На этом многие останавливаются, не пытаясь ответить на вопрос, почему происходит такое искажение.

Ответ заключается в том, что контроллер дисководов пишет информацию на диск битами и прекращение операции форматирования, то есть записи на дискету символа-заполнителя, прерывает посередине за-

пись очередного байта. Если учесть, что двоичное представление 4Eh = 01001110, то получается следующая картина (рис. 2).

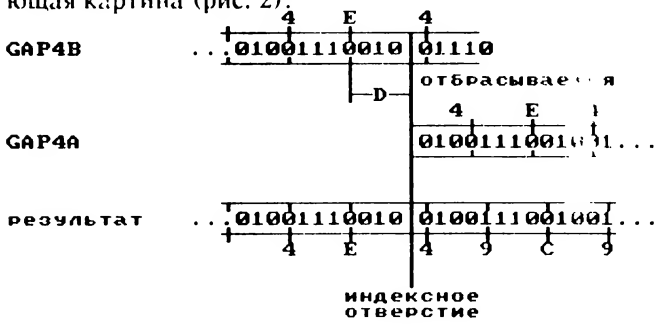


Рис. 2

Если обозначить через D количество битов, оставшееся от кода 4E при обрыве операции форматирования, а через S — значение первого байта после серии 4E, то можно определить следующую зависимость S от D (табл. 1).

Таблица 1

D, биты	1	2	3	4	5	6	7
S (hex)	27	53	49	44	4A	4D	4

Как видно из таблицы 1, при D=7 значение S совпадает со значением символа-заполнителя в GAP4B. В этом случае необходимо ориентироваться на следующий байт (обозначим его S'), значение которого должно быть равным 9Ch.

Тем не менее, в ряде случаев значения S будут отличаться от перечисленных в таблице. Хотя основная причина возникновения других значений и в этом случае все та же — обрыв операции форматирования, механизм, порождающий другие значения, несколько более сложен и основан на явлении, называемом смещением синхронизации. Это явление возникает из-за способа записи информации на дискету, при котором на представление одного бита информации на дискете отводится два бита. Этот способ известен под названием модифицированной частотной модуляции. Первый в паре битов называется битом тактовых импульсов (clock-bit), а второй — битом данных (data bit). Значения битов формируются следующим образом:

- если бит данных равен 1, то стоящий перед ним бит тактовых импульсов равен 0;
- если бит данных равен 0, то в случае, если предыдущий бит данных был равен 1, бит тактового импульса также равен 0. Однако если предыдущий бит данных равен 0, то бит тактового импульса равен 1.

Проиллюстрируем сказанное на примере символа-заполнителя 4Eh, стоящего в середине последовательности. Эта оговорка важна для правильного определения значений крайних в байте тактовых битов. На рис. 3 представлена последовательность импуль-

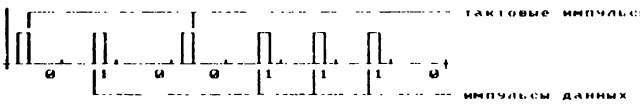


Рис. 3

ELITSoft

Пользователям вычислительной техники Акционерное общество "ЭЛИТ СОФТ" предлагает

Агрегаты Бесперебойного Питания (Uninterruptable Power Source)

Основные характеристики

- Мощность 900 ВА
- Время автономной работы 40 мин.
- Время переключения 5-7 мсек.
- Рабочий диапазон 180-240 В
- Гарантия 1 год
- Послегарантийное обслуживание

В конструкции предусмотрены

- Фильтрация сетевых помех
- Звуковая и световая сигнализация
- Индикация перегрузки
- Индикация уровня заряда батарей
- Герметичные, необслуживаемые аккумуляторные батареи

Поставка и оплата

Поставка со складов в Москве, Санкт-Петербурге, Чите, Рязани.
Любая удобная Вам форма оплаты.

Адрес: 105215 Москва, 11-я Парковая, 44, корп. 1

Телефон:
секретариат: (095) 965-35-29, 532-96-14
отдел сбыта: (095) 965-35-28, 965-35-30
Факс (095) 965-35-28, 965-35-30
Телетайп 205211 КТУР

Мы продаем компьютеры, модемы и факс-модемы производства США!

сов, считываемых с дискеты, и соответствующая им трактовка битов данных.

Как уже было отмечено, каждый информационный бит кодируется двумя битами на дискете, поэтому можно сказать, что информационный бит состоит из двух полубитов. Причем «кто есть кто», то есть какой полубит является тактовым, а какой битом данных, «знает» схема синхронизации. Если она «ошибается», возникает смещение синхронизации.

Теперь можно объяснить, почему возникают другие значения, отличные от описанных в таблице 1. Обрыв операции форматирования происходит после записи произвольного числа полубитов, которое может быть нечетным. Значения S для нечетного числа полубитов приведены в табл. 2.

Таблица 2

D, биты	0.5	1.5	2.5	3.5	4.5	5.5	6.5	7.5
S (hex)	90	48	68	54	49	4C	4E	4F

Экспериментальная проверка данных, приведенных в таблицах 1 и 2, показала, что на границе перехода GAP4B в GAP4A возможно появление других, отличных значений. Исследование этого явления, проведенное с помощью платы OPTION BOARD, позволило уточнить модель операции форматирования, которую теперь можно описать следующим образом. При заполнении поля GAP4B в момент обнаружения индексного отверстия контроллер прекращает текущую операцию и сбрасывает на диск содержимое внутреннего буфера. Если буфер к моменту прохождения индексного отверстия пуст, наблюдается ситуация, описываемая таблицами 1 и 2. Но если буфер содержит некоторые данные, то они сбрасываются на диск хаотическим образом.

Чтобы при измерении длины избежать этой ситуации, а также случая, когда D=0 (то есть когда невозможно найти границу между GAP4A и GAP4B), было предложено ориентироваться на границу между GAP4A и полем синхронизации, заполненным нулями. Если рассматривать это поле способом, описанным ранее, то есть как содержимое последнего большого сектора, то его легко выделить по последовательности нулей или, если сдвиг произошел на нечетное число полубитов, по последовательности значений FFh. Методика измерения в данном случае остается той же самой — по значению байта на границе поля GAP4A и поля синхронизации. Поиск граничного байта осуществляется по следующему алгоритму. Сканируется содержимое сектора до тех пор, пока не будет обнаружен байт со значением 00h или FFh. После этого необходимо убедиться, что это действительно поле синхронизации, путем проверки следующего байта, который должен совпадать с текущим. В противном случае необходимо продолжить сканирование. Когда поле синхронизации обнаружено, по значению предыдущего байта определяется D. Соответствие между ними определятся по табл. 3.

На основании полученных данных можно сделать вывод о том, что можно однозначно определить длину дорожки с точностью до полубита. При этом можно применить следующую последовательность действий:

Шаг 1. Отформатировать дорожку на K секторов с межсекторным расстоянием GAP3 и кодом длины секторов Nf так, чтобы последний сектор имел в заголовке большой код длины (порядка 4...6). Считать содержимое этого сектора.

Шаг 2. Сканировать содержимое сектора до тех пор, пока не будет обнаружено поле синхронизации по значению 00 или FF. Вернуться на байт назад к граничному байту S. Обозначим его смещение в рассматриваемом секторе через L_s.

Шаг 3. Определить по таблице 3 соответствующее значение D. Для значений, отмеченных в таблице звездочкой, значение L_s увеличивается на 1.

Шаг 4. Вычисляем длину дорожки L_t в полубитах по формуле

$$L_t = (125 + (60 + GAP3 + 128^{Nf} * 2) * (K - 1) + L_s) * 16 + D$$

Здесь 125 — расстояние от начала поля синхронизации до данных первого сектора в байтах; 60 — расстояние от начала заголовка сектора до начала данных в байтах.

В заключение хотелось бы сделать одно небольшое замечание для тех, кто решил воспользоваться данным способом измерения. Как чтение всей дорожки, так и чтение последнего сектора возможно только при прямом программировании контроллера дисководов, поскольку 13h прерывание в данном случае бесполезно. Поэтому прежде самого измерения Вам будет необходимо научиться работать с контроллером напрямую. Некоторые начальные сведения об этом можно получить в [2] и [3], ну а дальше необходимо изучать листинг BIOS. ■

Литература

1. **Защита программного обеспечения; Под ред. Д.Гроувера.– М.: Мир, 1992.**
2. **А.В.Спесивцев, В.А.Вегнер, А.Ю.Крутяков, В.В.Серегин, В.А.Сидоров. Защита информации в персональных ЭВМ.– М.: Радио и связь, 1992.**
3. **Р.Джордейн. Справочник программиста персональных компьютеров типа IBM PC, XT и AT.– М.: Финансы и статистика, 1992.**

Вниманию организаций и частных лиц!

Рекламное агентство компьютерного журнала «Монитор» приглашает к взаимному сотрудничеству по представительству интересов в городах России (вне Москвы), государствах СНГ и ближнего зарубежья (сбор рекламы и размещение ее в местной прессе, проведение рекламных кампаний и так далее).

С предложениями обращаться по телефонам:

отдел рекламы: (095) 903-46-26
редакция: (095) 238-16-22
факс: (095) 238-32-78

Таблица 3

D, п/бит	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S (hex)	90	23	7F	80	3F	C0	1F	E0	0F	70	87	38	43	9C	21
	*	*													

Копейка рубль бережет, или о том, как денежки счет любят

В настоящее время в Москве проблемы с программным обеспечением для банков практически решены. В лидеры вырвалось несколько крупных программных систем, принятых большинством коммерческих банков. Поскольку системы функционально почти одинаковы, основная конкурентная борьба идет за качество предоставляемых услуг.

В этой небольшой статье мы расскажем об особенностях только одной системы — программного комплекса «КОМБАНК», разработанного московской фирмой «ИНФА». Надеемся, что наши читатели, которые в своей повседневной деятельности так или иначе связаны с банками, почерпнут для себя что-то новое.

Итак, какие основные проблемы коммерческого банка решаются сегодня при помощи вычислительной техники? Прежде всего, это ведение операционного дня банка, безусловно, в многопользовательском режиме, что позволяет разворачивать специализированные рабочие места в необходимом количестве. Это удобно как для банка, так и для его клиентов.

И вот тут-то возникает первая сложность. НЕТ у нас в стране двух банков, которые похожи как «близнецы-братья». Стало быть, каждая система по-своему неповторима, уникальна. Можно, конечно, выдать заказчику коробку (или две) с дискетами и предоставить ему вариться в собственному соку.

Но фирма «ИНФА» избрала другой путь. В результате двухлетней работы был создан комплекс, состоящий из набора автоматизированных рабочих мест специалистов банка, которые связаны между собой информационными потоками и обслуживают единую базу данных. При этом каждому рабочему месту администратором системы могут быть назначены те или иные задачи (сегодня их уже более двух сотен), которые полностью учитывают особенности работы конкретного банка. Фирма «ИНФА» не просто продает программное обеспечение для банка, но и обеспечивает его внедрение на месте эксплуатации. Специалисты фирмы прилагают максимум усилий, чтобы внедрение нового программного комплекса **НЕ ПОТРЕБОВАЛО**:

- переучивания персонала (сохраняются даже привычные печатные формы),
- изменения привычного графика работы (ведь это неудобно клиентам),
- мучений при первоначальной установке системы и последующем расширении или корректировке ее функциональных возможностей.

Небольшая рекламная пауза.

Внедрение системы «КОМБАНК» не приводит к временному параличу деятельности банка при ее установке. Пользователи имеют возможность плавно перейти на новый комплекс, не теряя уже приобретенных навыков. Гибкая настройка пользовательского интерфейса позволяет максимально защитить пользователей — бухгалтеров, экономистов, операционистов — от информационной перегрузки, к которой привычны программисты.

Администратор системы имеет возможность на каждое рабочее место устанавливать только те задачи, которые необходимы для работы конкретного специалиста, а при необходимости изменять список назначений во время работы системы. Например, можно назначить операционисту Иванову следующие задачи:

- ввод платежных документов,
- работа с кассовыми документами,
- ведение документов по лицевым счетам,
- просмотр и изменение процентных ставок,
- обработка карточек со сканированными подписями клиентов банка,
- обработка документов, поступивших по системе «клиент-банк» и так далее.

— Стоп, стоп, стоп... — скажет читатель. — Что это за система «клиент-банк»? Приеду я, стало быть, в банк, и мне еще к какой-то системе надо подключаться?

А это, уважаемые читатели, вторая особенность системы «КОМБАНК». Уже сегодня клиенты банка, оснащенного системой фирмы «ИНФА», имеют возможность осуществлять расчеты с банком, не выходя из офиса. С помощью модема Вы можете передать в банк все необходимые платежные документы, достоверность которых заверяется цифровой подписью. При этом диагностируются все ошибки (неверно указанный лицевой счет, код МФО и так далее), после чего документ передается операционисту, а клиент получает квитанцию. Стоимость экономии драгоценного времени и нервной системы бухгалтера малого (или крупного) предприятия составляет всего 70$!

Что происходит с «платежкой» дальше? В Москве работа идет следующим образом. Деньги автоматически раз в сутки переводятся по тому же модему в РКЦ банка, который связан в единую сеть межцентрового обмена. В результате платеж клиента уже на следующий день оказывается у адресата (при условии, что банк адресата также в состоянии автоматически поддерживать обмен с РКЦ).

Но вернемся к программному обеспечению для банков. Предприниматели знают, что «поменять банк сложнее, чем поменять жену», а вот как обстоят дела с самой математикой для банка? Проблем возникает наверняка не меньше. А если речь идет об установке системы «с нуля»? В этом случае «КОМБАНК», пожалуй, не имеет себе равных. Поскольку проект разработан фирмой «ИНФА» с участием ведущих специалистов ЦБР, которые проявили понимание проблем и желание полностью автоматизировать всю банковскую деятельность в стране, при установке системы осуществляется ее первоначальная загрузка не только нормативно-справочной информацией, но и восстановленным по данным ЦБ содержанием счетов, проводок и прочих документов данного банка. Причем данные загружаются на месте установки по телефонному каналу.

В заключение приведем несколько технических подробностей.

Система реализована на FOXPRO 2.0 и функционирует в среде NOVELL 2.20, 3.11. Испытания, проведенные в КБ «Пресня-Банк» г. Москвы (средний оборот 4-5 тысяч документов в день, 25 рабочих мест), показали, что трафик в сети не превышает 15-20%. Для обеспечения межбанковского обмена данными используется система «Астра» разработки фирмы «ИНФА», обеспечивающая уверенную передачу данных в автоматическом режиме по телефонному каналу.

Более подробно о системе «КОМБАНК» Вам всегда расскажут в фирме «ИНФА».

Телефоны: (095) 176-1249, 176-0928, 306-6310 (по всем телефонам добавочный 24), факс 176-7998. ■

Средства сетевых СУБД для восстановления баз данных

Е. Морозов,
Е. Федорова

Повышение надежности автоматизированных информационных систем (АИС), базирующихся на локальных вычислительных сетях (ЛВС), является одной из актуальных проблем современной информатики. В ее основе лежит другая проблема — проблема организации восстановления баз данных (БД) в АИС.

Восстановление БД может обеспечиваться разными средствами, основные из которых поддерживаются системами управления базами данных (СУБД). Настоящая работа посвящается рассмотрению этих средств.

Проблема восстановления БД заключается в разработке и организации использования средств и методов, обеспечивающих сохранение целостности данных при нормальной работе АИС и воспроизведение целостности данных в случае ее нарушения при возникновении в АИС аварийных ситуаций.

Под целостностью данных понимается их согласованное (непротиворечивое) состояние, адекватно и полно отражающее состояние определенной предметной области.

Нарушение целостности данных может быть обусловлено различными причинами: сбоями и отказами аппаратуры, ошибками в программном обеспечении, неправильными действиями пользователей или операторов, а также быть результатом одновременного изменения несколькими пользователями одних и тех же данных (в табл. 1 приведен пример, поясняющий последнюю ситуацию).

Большинство повреждений данных обычно выявляется аппаратно-программными средствами контроля при отказе АИС, например, при аварийном завершении программ, отказе магнитного диска (МД), отказе вычислительной машины, аварийном завершении или зависании операционной системы (ОС) или СУБД, отключении питания.

Некоторые ошибки (в основном семантического характера) выявляются только самим пользователем при получении неверных результатов работы АИС или администратором БД после тестового контроля БД специальными программами.

Для сохранения целостности данных при работе с ними нескольких пользователей служит механизм блокирования. Сущность блокирования заключается в том, что в случае изменения данных одним из пользователей они объявляются системой недоступными до конца изменений для других пользователей.

Механизм блокирования обеспечивает только сохранение целостности данных, но не обеспечивает их восстановление в случае разрушения.

Существуют различные способы для восстановления БД. Простейшим из них является использование резервных копий БД, создаваемых через определенные промежутки времени. При отказе системы копия БД загружается и производится повторное внесение в нее всех изменений,

Таблица 1

Нарушение целостности данных вследствие их одновременного изменения несколькими пользователями

Событие	Программа	Действие	Результат
1		Читает запись	Запись содержит значение 2
2	В	Читает запись	Запись (все еще) содержит значение 2
3	А	Добавляет к значению 3 и пишет запись в БД	Запись содержит значение 5
4	В	Добавляет к значению 4 и пишет запись в БД	Запись содержит неверное значение: 6 вместо 9

выполненных с момента получения копии до момента отказа.

Однако этот способ сопряжен с большими затратами времени на восстановление и поэтому используется только при очень серьезных повреждениях БД.

В большинстве случаев используются другие способы восстановления, в основе которых лежит концепция организации в АИС транзактной обработки данных. Под транзакцией понимается единица работы над БД, которая переводит ее из одного целостного состояния в другое целостное состояние. Такое определение позволяет трактовать транзакцию и по-иному, а именно как единицу работы по восстановлению БД [1]. Обычно транзакция, включающая в себя набор логических операторов или процедур по модификации БД, ограничивается операторами НАЧАТЬ (begin) и ЗАВЕРШИТЬ (commit).

Ключевым моментом транзактной обработки является то, что при нормальной работе системы в БД вносятся все результаты операции, предусмотренные в рамках транзакции, или ни один из них в случае возникновения аварийной ситуации.

Таким образом можно избежать глобальных восстановлений БД и повтора больших наработок системы, заменив их восстановлением изменений, внесенных в рамках одной или нескольких транзакций.

Реализуется эта идея с помощью механизма блокирования и ведения журнала.

Процедуры восстановления и объем работ при восстановлении в значительной степени определяются тем, какие порции данных блокируются: файлы, записи или отдельные поля записей. В большинстве случаев это отдельные записи. В начале выполнения транзакции используемые ею данные блокируются, в конце — разблокируются.

Журналы представляют собой файлы, хранящие сведения обо всех изменениях, производимых над БД. Обычно в них фиксируются идентификаторы пользователей, время внесения изменений, образы записей БД до и после изменений (образы «до» и «после»), метки начала и конца транзакций [2].

Реализация основного принципа транзактной обработки — недопущение незавершенных корректировок в БД в рамках транзакции при возникновении аварийных ситуаций — обеспечивается средствами автоматического отката транзакций.

Последнее достигается ведением журнала транзакций. Перед выполнением любой модификации БД в журнал заносятся копии старых версий затронутых частей БД. При нормальном завершении транзакции журнальная информация не используется. В случае возникновения аварии и при рестарте системы эта информация служит для ликвидации результатов незавершенной транзакции (для отката транзакции к ее началу).

Примеры реализаций средств обеспечения целостности и восстановления

Сетевые СУБД обеспечивают возможность одновременной работы с БД многих пользователей для просмотра или корректировки данных. При этом традиционная схема работы предполагает, что все файлы БД хранятся в файл-сервере. По запросам пользователей требуемые файлы передаются на их рабочие станции, где и обрабатываются. Коллективное использование файлов приводит к необходимости применения средств блокирования [3], [4].

В СУБД Paradox существует возможность отслеживания всех файлов и записей, находящихся в использовании на каждой рабочей станции, а также типов действующих в сети блокировок.

Paradox поддерживает множество типов блокировок данных, основными из которых являются полная блокировка, блокировка от записи, предохраняющая блокировка от записи, предохраняющая полная блокировка.

Полная блокировка является наиболее ограничивающей. Она дает пользователю полный доступ к таблице или другому информационному объекту (форме или отчету), полностью предотвращая доступ всех других пользователей к тому же объекту по любой причине.

Блокировка от записи позволяет другим пользователям иметь доступ к используемой таблице или другому объекту, но не позволяет им изменять его структуру или содержание.

Предохраняющая блокировка от записи предохраняет объект от наложения на него другим пользователем полной блокировки или блокировки от записи. Эта блокировка гарантирует, что другие пользователи не помешают выполнить операции, требующие модификации объекта.

Предохраняющая полная блокировка предохраняет объект от наложения на него другими пользователями полной блокировки, эта блокировка обеспечивает максимальный уровень совместного использования объекта. Другие пользователи могут выполнять операции, которые требуют наложения любого типа блокировки, кроме полной. Третья и четвертая блокировки ограничивают возможность наложения первых двух.

Paradox блокирует объекты разумно. В каждой ситуации он автоматически применяет наименее ограничивающую блокировку в соответствии с выполняемой пользователем операцией.

Наряду с автоматическим блокированием имеется возможность явного блокирования объектов самим пользователем. Последнее может потребоваться ему, когда он точно знает, что определенная блокировка необходима, и желает быть уверенным в ее осуществлении.

В отличие от СУБД Paradox СУБД Advanced Revelation никогда не блокирует автоматически весь файл. Эта СУБД позволяет обрабатывать файл не как единый блок данных, а как определенное количество защищенных групп записей, то есть блокируется только небольшая группа записей на время в несколько машинных тактов. Такой способ блокировки позволяет другим пользователям работать с незаблокированной частью БД. Следствием этого является повышение производительности и оперативности обработки информации. При записях в файл привилегии автоматически получает первый пользователь.

СУБД R:base for DOS [9] имеет утонченный механизм, обеспечивающий автоматическое выполнение наиболее слабой блокировки. Она позволяет пользователям корректировать одну и ту же запись, пока они работают над непересекающимися частями этой записи.

Если два пользователя обращаются к одной и той же записи, то приоритет на сохранение изменений, сделанных в записи, имеет первый пользователь. Второй пользователь будет предупрежден о том, что в записи были сделаны изменения (и кем), и он может затем решить, принять ли ему изменения, сделанные в записи первым пользователем, или выполнить новую корректировку.

Это позволяет максимизировать коллективный доступ к записям базы данных и в то же время обеспечить целостность данных, что, однако, приводит к уменьшению производительности системы, так как R:base for DOS вынуждена постоянно выполнять проверки наличия конфликтов в значениях записей БД.

Механизмы блокирования реализованы во всех сетевых СУБД, тогда как транзактная обработка присуща только тем, которые поддерживают схему обработки данных «клиент-сервер». К ним, в частности, относятся СУБД MDBS, ORACLE SERVER, Advanced Revelation, PROGRESS, Btrieve IV [5], [6], [7], [8].

Программное обеспечение СУБД для серверов тесно смыкается с сетевыми ОС. Характерным примером этого является система управления записями Btrieve, входящая в состав ОС NetWare. Рассмотрим ее возможности в плане восстановления данных более подробно.

Для обеспечения целостности данных Btrieve использует механизм блокирования. Возможно блокирование всего файла, одной или нескольких записей в файле. При этом различают блокирование с ожиданием и без ожидания.

Блокирование с ожидание делает невозможным возврат управления прикладной программе пользователя до тех пор, пока блокировка данных не будет успешно установлена. То есть если данные были ранее заблокированы с другой станции, то прикладная программа пользователя будет ждать ее освобождения.

Блокирование без ожидания немедленно возвращает управление, даже если блокируемые данные удерживаются другим пользователем. Последнее сообщается прикладной программе через соответствующий код возврата.

Btrieve обеспечивает автоматическое восстановление данных с помощью системы поддержки транзакций Transaction Tracking System (TTS) или с помощью системы Pre-imaging.

Средства восстановления TTS являются основными, если пользователь в своих прикладных программах использует логику транзакций.

Транзакции могут быть двух видов: явные и неявные. **Явная** транзакция начинается выдачей команды TTSBeginTransaction (начать транзакцию), а заканчивается выдачей команды TTSEndTransaction (закончить транзакцию).

Неявная транзакция начинается автоматически, когда прикладная программа первый раз блокирует запись файла, помеченного специальным флагом как Transactional (транзакционный). Флаг устанавливается автоматически при инсталляции Btrieve или после инсталляции с помощью специальной утилиты.

При этом для использования возможности TTS по автоматическому восстановлению (автооткату) транзакций должны быть выполнены 2 условия:
- TTS должна быть активна в файл-сервере;
- защищаемые файлы должны быть помечены как Transactional.

Обработка транзакции происходит следующим образом:

1. Прикладная программа пользователя выдает команду на запись файла.
2. Новая информация пользователя запоминается в кэш памяти файл-сервера. Исходный файл пользователя на файл-сервере не обновляется.
3. На файл-сервере сканируется исходный файл на предмет поиска старой информации, подлежащей обновлению. Эта информация копируется в кэш-память.
4. Старая информация из кэш-памяти пишется в файл транзакций, определяемый при генерации системы.
5. Приложение пользователя выдает команду окончания транзакции, после чего новая информация из кэш-памяти переписывается в файл. Заканчивается транзакция разблокированием всех записей.

Если транзакция по какой-либо причине прерывается, содержимое файла транзакций используется для автоматического восстановления старой информации, то есть для отката транзакции к ее началу.

Информация файла транзакций используется и для восстановления после аварий системы. При этом в большинстве случаев восстановление происходит, как только пользователь открывает свои файлы после аварии.

Если условия поддержки транзакций не выполняются, то есть файлы не помечены как Transactional или TTS не активна, Btrieve обеспечивает восстановление с помощью процесса Pre-imaging. В случае аварийного завершения операции по обновлению данных Btrieve автоматически восстанавливает их к состоянию, которое было перед выдачей запроса на обновление.

Для обеспечения процесса Pre-imaging Btrieve создает временный файл сразу, как только прикладная программа изменяет пользовательский файл. Причем каждому пользовательскому файлу соответствует свой временный файл pre-image. Этот временный файл используется, чтобы отследить изменения пользовательского файла в рамках запроса.

В течение операции изменения Btrieve запоминает новый образ записи в кэш-буфере, а старый образ — в pre-image буфере, затем записывает pre-image буфер во временный файл и только потом вносит изменения, то есть запоминает новый образ записи в пользовательском файле.

Таким образом, запоминание старого образа записи во временном файле всегда предшествует изменению в пользовательском файле. Созданный временный файл pre-image не уничтожается до тех пор, пока пользовательский файл не будет успешно закрыт.

Наряду с указанными средствами блокирования, управления транзакциями и pre-image Btrieve поддерживает три разных метода управления конкуренцией, предназначенных для разрешения конфликтов, которые могут случиться, когда два разных пользователя со своих рабочих станций пытаются изменить или удалить одни и те же записи в одно и то же время. Эти методы ориентированы на ситуации:

а) когда конкурирующие между собой прикладные программы используют логику транзакций,

б) не используют ни логики транзакций, ни механизма блокирования,

в) когда одна из конкурирующих программ использует логику транзакций, а другая не использует никакого механизма защиты.

Рассмотрим первый метод. Он представляет собой классическое управление транзакциями.

Всякий раз как прикладная программа взаимодействует с файлом в пределах транзакции, Btrieve поддерживает доступ к файлу до тех пор, пока закончится или прервется транзакция. Другие станции могут читать файл при условии, что чтение не выполняется в пределах транзакции. Иначе в пределах транзакции никто не может получить доступ к файлу.

Btrieve обычно выполняет блокирование с ожиданием. Если 2 станции пытаются получить доступ к одному и тому же файлу в пределах транзакции, Btrieve задерживает вторую станцию до тех пор, пока первая станция не закончит свою транзакцию.

Таблица 2 показывает, как две станции взаимодействуют, пока работает одна и та же прикладная задача, которая читает и модернизирует два файла в пределах ожидающей транзакции. Шаги пронумерованы для того, чтобы показать порядок, в котором наступают события.

Таблица 2

Управление транзакциями

Действия программы станции 1	Действия Btrieve	Действия программы станции 2
(1) Открытие файла 1		
		(2) Открытие файла 1
(3) Открытие файла 2		
		(4) Открытие файла 2
(5) Начало транзакции		
		(6) Начало транзакции
(7) Чтение файла 1		
		(8) Чтение файла 1
	(9) Возврат программе станции 2 кода ожидания	
(10) Чтение файла 2		
(11) Модификация файла 1		
(12) Модификация файла 2		
(13) Окончание транзакции		
	(14) Разрешение завершения чтения	
		(15) Чтение файла 2
		(16) Модификация файла 1
		(17) Модификация файла 2
		(18) Окончание транзакции

Недостатком этого метода управления конкуренцией является возможность возникновения взаимной блокировки двух программ (тупика).

Таблица 3 демонстрирует, как две станции могут стать взаимно блокированными. Станции используют две различные программы, которые обращаются к одному и тому же файлу в разном порядке без ожидания транзакции.

Таблица 3

Пример взаимной блокировки

Действия программы станции 1	Действия Btrieve	Действия программы станции 2
(1) Открытие файла 1		
(2) Открытие файла 2		
		(3) Открытие файла 1
		(4) Открытие файла 2
(5) Начало транзакции		
		(6) Начало транзакции
(7) Чтение файла 1		
		(8) Чтение файла 1
(9) Чтение файла 2		
	(10) Задержка программы станции 1, пока станция 2 блокирует файл 2	
		(11) Чтение файла 1
	(12) Задержка программы станции 2, пока станция 1 блокирует файл 1	

Рассмотренные средства сетевых СУБД для восстановления БД показывают, что их «арсенал» достаточно мал. При этом следует учесть, что концепция транзактной обработки, несмотря на всеобщее признание, до сих пор не получила в программных системах на базе ЛВС широкого распространения. Все это означает, что проблема восстановления в целом еще не нашла своего решения. ■

Литература

1. Алиев А.А. Контрольные точки и откат-восстановление в распределенных системах// Управляющие системы и машины.– 1988.– № 4.
2. Морозов Е.А. Обзор способов организации и использования журналов современных СУБД// Прикладная информатика.– 1991.– Вып. 17.
3. Холл Ченел, Мерлайни Найсери. Особенности работы с базами данных в сети: обозрение сетевых характеристик СУБД// КомпьютерПресс.– 1989.– № 1.
4. Когаловский М.Р. Технология баз данных на персональных ЭВМ.– М.: Финансы и статистика, 1992.
5. Проничева Д. Серверы баз данных// КомпьютерПресс.– 1991.– № 11.
6. Беленькая М.Н., Карпова Т.В., Самойлов А.И. СУБД для локальных вычислительных сетей на базе персональных компьютеров// Журнал д-ра Добба.– 1991.– № 3.
7. Каталог продуктов фирмы Novell// КомпьютерПресс.– 1992.– № 4.
8. Астафьева С., Беленькая М., Бухман А., Карпова Т., Самойлов А. СУБД Btrieve в среде Novell// КомпьютерПресс.– 1991.– № 8.

Резидентный «будильник» на Turbo Pascal 5.0

А. Шибенко

Виктору Белею с благодарностью

Описываемая программа была создана как составной элемент пакета «Электронный секретарь», включающего, кроме прочих, функции ежедневника с памяткой. Первоначально занесенные в памятку сообщения могли выводиться на экран лишь в случаях, когда в этот момент пользователь работал непосредственно с «секретарем». Однако требования рынка заставили взяться за доработку. Определился следующий круг задач для нового модуля:

- резидентное размещение в памяти;
- чтение дискового файла, в который «секретарь» заносит задаваемое пользователем время выдачи сообщений и напоминаний;
- оповещение пользователя в доступной ему форме при наступлении времени выдачи сообщения;

Принципиальная возможность написания подобной программы на ассемблере или С сомнений не вызывала, но положение осложнялось весьма поверхностным знанием автором первого и наличием под рукой лишь PC/XT, на винчестер которой емкостью 20 Мб какая-либо версия второго просто не помещалась. Однако имелся Turbo Pascal 5.0, а в [1] удалось найти описание и листинг TSR программы на этом языке. Лед тронулся.

При написании резидентных программ основной проблемой считается преодоление нереентерабельности ДОС. В [2], [3] «безопасное» состояние ДОС, во время которого можно пользоваться ее функциями, определяется с помощью прерывания 28h. В описываемой программе для решения второй задачи также используется собственный обработчик этого прерывания (процедура SetSignal). Логика его работы следующая. Чтение файла происходит в двух случаях:

- изменился его размер, то есть пользователь ввел еще одно или несколько значений времени выдачи сообщения;
- закончилась выдача очередного сообщения и необходимо определить время выдачи следующего.

Так как времена в файле не упорядочены, в цикле просматриваются все значения и выбирается наименьшее из не превышающих установленного в системе.

Системное время берется из ячеек 0:46Ch–0:46Fh области данных BIOS, где хранится счетчик импульсов таймера, обновляемый 18,2 раза в секунду [4].

Выдача сообщения осуществляется обработчиком прерывания 1Ch (процедура Alarm). При совпадении времени, определяемого процедурой SetSignal, с системным включается динамик (о том, как программировать микросхему таймера 8253/8254, можно прочитать в [4], [5]) и, если дисплей работает в одном из текстовых режимов, в правом нижнем углу экрана выводится красный прямоугольник. При этом используется запись в видеопамять по абсолютным адресам. Через 1,5 секунды динамик выключается и восстанавливается исходное изображение на затертой прямоугольником части экрана.

Для резидентной программы важно исключить возможность ее повторной установки. В [2] это делается путем перехвата прерывания 2Fh. Однако реализовать подобное на Pascal не удалось. Даже простейшая процедура вида:

```
Procedure Intr2F(...); interrupt;
Begin
   Inline($9C);   { pushf }
   Intr2FOld;     { вызов исходного
         обработчика прерывания 2Fh }
End;
```

приводила к зависанию системы при попытке сделать ее резидентной, хотя нормально работала в качестве нерезидентного обработчика. Причину подобного поведения установить так и не удалось (буду благодарен тому, кто сможет мне объяснить этот нюанс), поэтому было сделано вот что.

Исходный обработчик прерывания 1Ch содержит лишь инструкцию iret и через регистры ничего не возвращает. А в процедуре Alarm при обращении к ней со значением параметра AX, равным 10, оно изменяется на 255. Теперь достаточно в начале нерезидентной части программы вызвать прерывание 1Ch со значением регистровой переменной REG.AX, равным 10, проверить ее выходное значение и закончить работу программы, если оно равно 255.

Остальное вполне понятно из листинга.

Получившаяся программа довольно успешно решает возложенные на нее задачи, занимая в памяти примерно 10 Кб. ■

Литература

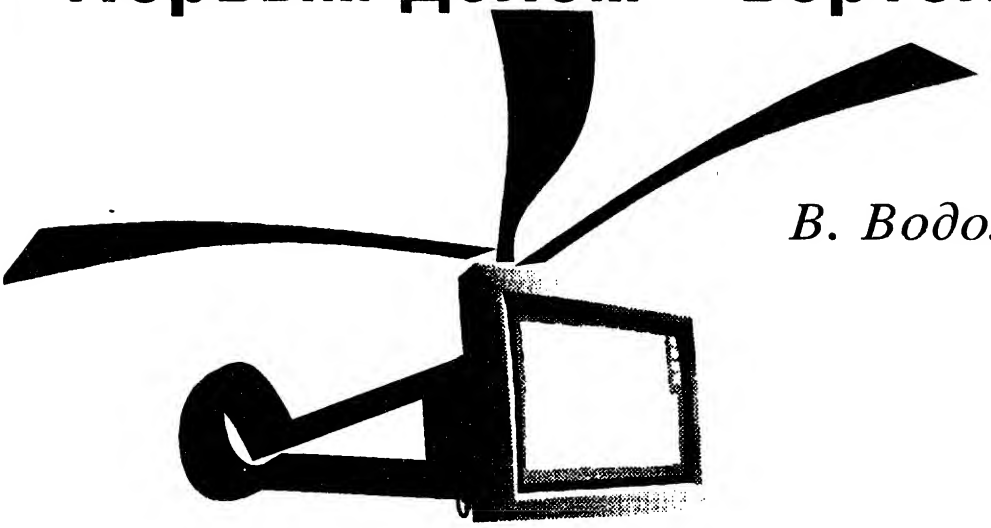
1. М. Остринский. Создание резидентных программ и работа с прерываниями// Монитор.– 1991.– № 1.
2. А. Рыскунов. Этот безумный, безумный, безумный мир резидентных программ// КомпьютерПресс.– 1992.– № 4, 5.
3. Э. Стивенз. Новый взгляд на TSR-программы// Журнал д-ра Добба.– 1991.– № 3.
4. Р. Джордейн. Справочник программиста ПК типа IBM PC XT/AT.– М.: Финансы и статистика, 1991.
5. Фролов А.В., Фролов Г.В. Аппаратное обеспечение IBM PC.– М.: Диалог-МИФИ, 1992.

Листинг

```
Program Rezident;
{
Резидентная программа "БУДИЛЬНИК".
Copyrigth (C) А.Н.Шибенко, 1993 г.
}
{$M 1024,0,0}
uses CRT,DOS;
type Ar22 = array[1..22] of byte;
var
    BOOL,READTIME      : boolean;
    FREQ               : word;
    EN1,EN2,EN3,                { буферы для обмена }
    EO1,EO2,EO3        : Ar22; { с экраном }
    TIM                : longint absolute 0:$046C;
                        { системное время }
    FSIZEOLD,TSIGNAL   : longint;
    PATH               : PathStr;
    REG                : registers;
    Int1C,Int28        : procedure;
    F                  : file of longint;

{F+}
Procedure
Alarm(FLAGS,CS,IP,AX,BX,CX,DX,SI,DI,ES,DS,BP:word);
        interrupt;
{
Обработчик прерывания 1Ch
подает звуковой сигнал и выводит в правом
нижнем углу экрана красный прямоугольник.
}
var
    E1 : Ar22 absolute $B800:3658;  { правый нижний }
    E2 : Ar22 absolute $B800:3818;  { угол экрана для }
    E3 : Ar22 absolute $B800:3978;  { цветного дисплея }
    B  : byte;

Begin
B:=Lo(AX);
if (TIM>=TSIGNAL) and (TIM<=TSIGNAL+24) then
                        { наступило время сообщения }
    begin
    REG.AH:=$F;
    Intr($10,REG);      { для определения режима дисплея }
    if not BOOL then    { сообщение еще не выводилось }
        begin           { включение звука }
        Port[$43]:=$B6;
        Port[$42]:=Lo(FREQ);
        Port[$42]:=Hi(FREQ);
        Port[$61]:=Port[$61] or 3;
        if (REG.AL<=3) or (REG.AL=7) then
                        { для текстовых режимов }
            begin       { запоминание экрана }
            EO1:=E1;
            EO2:=E2;
            EO3:=E3;
        end;
    end;
    if (REG.AL<=3) or (REG.AL=7) then
                    { для текстовых режимов }
        begin       { вывод напоминания в течение 1.5" }
        E1:=EN1;
        E2:=EN2;
        E3:=EN3;
    end;
    BOOL:=true;         { сообщение уже выводится }
    READTIME:=false;{ запрещение чтения нового
        времени до обработки очередного сообщения }
end;
if BOOL and (TIM>=TSIGNAL+25) then    { прошло 1.5" }
    begin
    BOOL:=false;            { окончен вывод сообщения }
    READTIME:=true;         { необходимо прочитать новое
                              время из файла }
    Port[$61]:=Port[$61] and 252; { выключение звука }
    REG.AH:=$F;
    Intr($10,REG);      { определение режима дисплея }
    if (REG.AL<=3) or (REG.AL=7) then
                    { для текстовых режимов }
        begin       { восстановление прежнего экрана }
        E1:=EO1;
        E2:=EO2;
        E3:=EO3;
    end;
end;
if B=10 then AX:=255;       { для проверки повторной
                              установки }
Port[$20]:=$20;             { end of interrupt }
End;

Procedure
SetSignal(FLAGS,CS,IP,AX,BX,CX,DX,SI,DI,ES,DS,BP:
        word); interrupt;
{
Обработчик прерывания 28h
читает неупорядоченный файл времен и опре-
деляет время выдачи очередного сообщения.
}
var
    FSIZE,TI : longint;

Begin
reset(F);
FSIZE:=FileSize(F);     { определение текущего размера
                          файла времени }
if (FSIZE<>FSIZEOLD) or READTIME then
            { изменился файл времени либо отработало }
    begin   { текущее сообщение }
    READTIME:=false;{ запрещение чтения нового
            времени до обработки очередного сообщения }
    FSIZEOLD:=FSIZE;    { сохранение текущего размера }
    TSIGNAL:=24*3600*19;
                        { фиктивное время > 24 часов }
    while not Eof(F) do begin
        read(F,TI);
        if (TI>TIM) and (TI<TSIGNAL) then TSIGNAL:=TI;
                        { в TSIGNAL <-
                        время очередного сообщения }
    end;
end;
close(F);
Inline($9C);    { pushf }
Int28;          { исходный обработчик 28h }
End;
{$F-}

Begin
REG.AL:=10;             { проверка наличия собственного }
Intr($1C,REG);          { обработчика 1C и исключение }
if REG.AL=255 then Halt; { повторной установки }
BOOL:=false;
READTIME:=true;
        { необходимо прочитать новое время из файла }
for TSIGNAL:=1 to 11 do begin
                        { заполнение буфера экрана }
    EN1[1+2*(TSIGNAL-1)]:=219;  { маской сообщения }
    EN1[2*TSIGNAL]:=4;
    EN2[1+2*(TSIGNAL-1)]:=219;
    EN2[2*TSIGNAL]:=4;
    EN3[1+2*(TSIGNAL-1)]:=219;
    EN3[2*TSIGNAL]:=4;
end;
TSIGNAL:=24*3600*19;    { фиктивное время > 24 часов }
PATH:=FExpand('Time.tim');  { путь к файлу времен }
assign(F,PATH);
FSIZEOLD:=0;            { фиктивный начальный размер
                          файла времени }
FREQ:=Round(1193180 div 500);
                        { частота звукового сигнала }
GetIntVec($1C,@Int1C);      { установка собственных
                              обработчиков }
SetIntVec($1C,Addr(Alarm)); { 1Ch и 28h }
GetIntVec($28,@Int28);
SetIntVec($28,Addr(SetSignal));
Keep(0);
End.
```

■

Первым делом — вертолеты...



В. Водолазкий

– Товарищ прапорщик, а крокодилы летают?
– Да вы что, рядовой Иванов, конечно же, нет!
– А вот товарищ капитан говорит, что летают...
– Ну разве что очень низко, над самой землей...

Старый армейский анекдот

Название игры:	*Comanche Maximum Overkill*
Фирма-разработчик:	*NovaLogic*
Процессор:	*80386 и выше (защищенный режим!)*
Оперативная память:	*не менее 4 Мбайтов*
Монитор:	*VGA 256 цветов*
Управление:	*клавиатура/«мышь»/ джойстик/ThrustMaster/ FlightStick*
Звуковые эффекты:	*PC Speaker/ Adlib/ Sound Blaster/ Roland*
Объем на диске:	*12 Мбайтов*

Как ни странно, но об авиационных имитаторах, называемых в просторечье «леталками», наш журнал еще ни разу не писал. Но мы постараемся восполнить этот пробел и начать хотим с новой игры, появившейся на рынке в 1992 году.

Прежде всего отметим достижения фирмы-разработчика. Небольшая и малоизвестная фирма NovaLogic анонсировала качественно новую технологию визуализации трехмерных поверхностей, получившую название VoxelSpace. По заявлению фирмы, VoxelSpace обеспечивает формирование до 48000 трехмерных полигонов за время смены кадра в реальном масштабе времени. При этом обеспечивается закраска каждого из полигонов с учетом пространственного положения источников освещения. Но за эту технологию приходится платить. Цена — используемый процессор (не ниже 80386) и оперативная память (не менее 4 Мбайтов). Причем процессор должен работать в защищенном режиме, а следовательно, Вам придется выгрузить из памяти 386MAX, EMM386 и им подобные.

Но если эти ограничения Вас не пугают, то имеет смысл попробовать свои силы в управлении вертолетом Comanche. Конечно, по увлекательности эта игра значительно уступает лидерам: F-117 и F-15, да и старый добрый LHX был продуман все-таки более тщательно. Вообще говоря, складывается впечатление, что выпуск Comanche преследует только цель заявить о достижениях NovaLogic в области машинной графики, а вовсе не стать рекордсменом по продажам компьютерной игры. Игра при всей своей внешней привлекательности, производящей впечатление как на игроков-любителей, так и на программистов-профессионалов, достаточно быстро приедается. Хотя, как говорится, на вкус и цвет...

Но давайте поговорим обо всем по порядку. Прежде всего, для того чтобы эффективно справляться с поставленными перед Вами задачами, Вы должны отчетливо представлять себе возможность техники, находящейся в Ваших руках. Применительно к имитаторам это означает, что Вы должны владеть не только определенными познаниями в материально-технической части того или иного самолета или вертолета, но также иметь некоторые познания в тактике применения того или иного вида оружия. Мы планируем поместить в нашем журнале ряд статей о типовых тактических ситуациях в имитационных играх и приглашаем прежде всего военных специалистов принять участие в работе нашего журнала. А пока мы попытаемся справится своими силами.

Что такое Comanche

Легкий тактический вертолет огневой поддержки RAH-66 Comanche сегодня находится еще в стадии разработки. Эта модель представляет собой дальнейшее развитие попыток создания вертолетов-невидимок (проект LHX). RAH-66 представляет собой легкий двухместный аппарат, предназначенный как для борьбы с наземными бронированными целями, так и для использования в качестве системы целеуказания и наведения в разведывательно-ударных комплексах дивизионного и корпусного звена сухопутных войск армии США. Известная операция «Буря в пустыне» только усилила и без того явно выраженную тенденцию Пентагона к возможно более широкому использованию высокоточного оружия. К чему это привело? Газета «Правда» обвинила американцев в низком боевом духе: солдаты, видите ли, умирать не хотят, все норовят во врага ракетой шваркнуть! А американцы снизили требования к боевой нагрузке вертолета огневой поддержки (!), что позволило уменьшить вес вооружения до 4 тонн (для сравнения: Apache — 6 тонн, Black Hawk — 8,5 тонн). Кроме того, крейсерская скорость RAH-66 составляет 185 км/час. Использование STEALTH технологии (не путать со STEALTH вирусами!) дает RAH-66 дополнительные преимущества при борьбе с войсковыми средствами противовоздушной обороны. Однако в дуэли «воздух-земля» достаточно важным фактором современного боя является время реакции. Пилот не успевает полностью отслеживать всю обстановку и адекватно на нее реагировать. Поэтому-то экипаж Comanche и состоит из двух человек: пилота и оператора прицельно-навигационной системы.

В игре обе функции придется выполнять Вам. Начнем с пилотирования.

Управление полетом

Чем обычно недоволен компьютерный пират? Прежде всего сложностью управления, невозможностью освоиться с игрой без документации. В этом плане игра фирмы NovaLogic — просто находка. Для управления игрой достаточно использовать дополнительную цифровую клавиатуру.

Самая важная клавиша «*». При ее нажатии вертолет переходит в режим автоматического огибания рельефа местности. Высота полета составляет 32 или 64 метра. Единственная проблема — инерционность системы управления, в результате чего Вы при пересечении, например, реки с большой долей вероятности врежетесь в противоположный берег.

Для того чтобы подняться повыше, нажмите клавишу «-». Чтобы снизиться — «+». Стрелки управления курсором определяют направление движения. Клавиши «0» и «.» на цифровой клавиатуре позволяют Вам управлять хвостовым винтом и совершать повороты, зависнув на месте.

И все! Это Вам не MegaFortress с несколькими десятками переключателей и даже не F-117, в котором, по крайней мере, необходимо убрать шасси и регулировать положение закрылков.

Более того, Вам необходимо только взлететь. После выполнения задания включается таинственный автопилот, который невредимым несет Вас через горы и долы. Оставшийся противник в знак уважения к Вашим заслугам прекращает огонь. Посадка в этой игре не предусмотрена. Более того, по сути говоря, не предусмотрен и полет как таковой. С самого начала выполнения задания Вы оказываетесь в самой гуще событий, в окружении противника. Необходимо только маневрировать и успевать метко стрелять.

Поиск жертвы

Прежде чем Вы нажмете на гашетку, необходимо наметить объект, подлежащий ликвидации. Противников много, но для выполнения поставленной перед Вами задачи не всегда необходимо уничтожать всех подряд. В некоторых случаях достаточно уничтожить всего пять-шесть объектов. В их поиске огромную помощь Вам окажет цифровая карта местности. Наиболее удобный режим работы индикаторов достаточно верно подмечен авторами игры и устанавливается по умолчанию: левый индикатор — общая карта местности, правый — телевизионный прицел.

Так вот, на цифровой карте Вы можете увидеть местоположение всех объектов:

- белый крест — Ваше местоположение;
- белая точка — стационарные объекты (нефтехранилища);
- желтая точка — подвижные наземные цели (танки, комплексы ПВО);
- красная точка — вертолеты противника;
- зеленая точка — Ваш напарник.

Объекты, которые являются целью Вашего задания, дружески подмигивают Вам на карте, но не забывают при первом удобном случае направить в Ваш адрес «приветственную делегацию».

Стрельба по мишеням

Итак, как мы уже говорили ранее, вертолет Comanche имеет не слишком мощное вооружение. Поскольку упор сделан на применение высокоточного оружия, для успешного выполнения задания Вы должны научиться с равным успехом пользоваться всеми видами предоставляемого Вам вооружения. В этом отношении Comanche выгодно отличается от F-117 и F-15, в которых большинство заданий можно выполнить с помощью самонаводящейся ракеты типа Maverick или AMRAAM.

Автоматическая 20 мм пушка Gatling Vulcan II. Трехствольная 20 мм пушка размещается внутри фюзеляжа вертолета, что позволяет уменьшить общую отражающую поверхность и обеспечить более высокую защищенность орудия от осколков и снарядов противника. Орудие размещается на гиростабилизированной платформе и сопряжено с навигационно-прицельной системой. Пушка обеспечивает эффективное поражение наземных целей на коротких и средних дальностях. Но чтобы поразить бронированный объект, например, танк, Вам необходимо истратить достаточно много снарядов. Темп стрельбы составляет примерно 1500 выстрелов в минуту, поэтому при непрерывной стрельбе Ваш боезапас истощится менее чем через минуту. Наиболее эффективно применение пушки против слабозащищенных объектов, в частности, нефтехранилищ. С переменным успехом можно стрелять и по вертолетам, однако в этом случае угол упреждения Вы должны устанавливать вручную.

70 мм неуправляемые реактивные снаряды «воздух-земля». Это отдаленный потомок наших «катюш». Исключительно эффективное средство! Достаточно впечатляюще выглядит расстрел звена вертолетов противника на встречном или пересекающемся курсе. Дальность стрельбы составляет около двух километров. Прицельно-навигационная система обеспечивает нацеливание только по азимуту. Угол места Вы должны устанавливать самостоятельно. Стрельба осуществляется прямо по курсу движения вертолета. Так как ракеты неуправляемые, то после запуска Вы не можете корректировать направление их полета. Все зависит от Ваших навыков и удачи.

Не забывайте о клавише «M». Даже если Вы выбрали другой вид оружия, нажатие на эту клавишу приведет к немедленному выстрелу парой неуправляемых ракет с обеих бортов. Вы можете использовать эту возможность, например, когда на Вас внезапно «лоб в лоб» выныривает вертолет противника.

Ракеты с лазерным наведением класса «воздух-земля» AGM-114 HellFire. Это универсальный вид оружия, который можно эффективно применять для поражения как наземных, так и воздушных целей. Дальность поражения Hellfire составляет свыше 8 км. Наведение осуществляется с помощью телевизионного прицела. Мощность боеголовки вполне достаточна, чтобы с одного попадания вывести из строя любую наземную, в том числе и бронированную цель. Но за это приходится платить. Для того чтобы поразить цель, Вам необходимо захватить ее телевизионным прицелом (клавиши TAB и Enter), запустить ракету и удерживать цель в прицеле до попадания. В случае если Вы потеряли цель из вида (как правило, из-за складок местности), Вы можете перенацелить уже летящую ракету на другой объект, находящийся в поле зрения.

Hellfire — это один из самых надежных видов вооружения Вашего вертолета. Не растрачивайте его на второстепенные цели. Например, для поражения нефтехранилищ достаточно одного-двух залпов из пушки. В то же время не следует забывать, что при необходимости Вы можете с успехом использовать эти ракеты и против вертолетов противника.

Ракеты с ИК головкой самонаведения AIM-92 Stinger. Об этих ракетах все мы давно знаем из телевизионных репортажей и газетных статей о событиях в Афганистане. Дальность действия ракет Stinger составляет от одного до двух километров. Головка самонаведения позволяет реализовать схему применения «выстрелил и забыл». Однако Stinger имеет более низкую эффективность, чем Hellfire, и для поражения наземных бронированных целей Вам потребуются три-четыре ракеты.

Вот и все вооружение, которое размещается на борту RAH-66. Однако Вы имеете возможность использовать и так называемые «внешние» системы оружия. Фактически Вы как бы включаетесь в небольшой разведывательно-ударный комплекс, собираемый командиром дивизии из подручных материалов. В таком комплексе «глазами» будет Ваш вертолет, а средством поражения могут быть либо артиллерийские дивизионы, либо Ваш напарник.

155 мм артиллерия и ракетные комплексы MLRS. Артиллерийская поддержка представляет собой наиболее мощный с огневой точки зрения вид оружия. Основная идея состоит в следующем. Бортовой компьютер вертолета постоянно определяет Ваше местоположение. Как только Вы захватили цель телевизионным прицелом, компьютер с помощью лазерного прицела/дальномера определяет ее относительные и абсолютные координаты. После этого данные могут быть переданы на КП артиллерии дивизии, откуда и поступит команда на поражение.

Из этого введения Вам необходимо сделать следующие выводы. Во-первых, после того как Вы передали целеуказание, можете переключаться на другие цели, а во-вторых, необходимо учитывать, что на прохождение команды и наведение орудий требуется время. Поэтому, если Вы пытаетесь передать артиллерии движущиеся объекты, то с большой долей вероятности удар придется по пустому месту. Противник тоже ценит маневр! В то же время вертолеты на заправке — отличный объект для применения артиллерии.

К сожалению, огневая поддержка предоставляется далеко не во всех задачах. Но уж если есть возможность ее использовать, не упускайте ее!

Напарник. В некоторых задачах Вы вылетаете на задание с напарником. Как уже указывалось, его вертолет отображается на цифровой карте зеленой точкой. Напарник вооружен ракетами типа Hellfire, но распоряжаетесь ими Вы.

Как правило, напарник располагается все время позади Вас, прикрывая хвост Вашего вертолета. Надо признать, что противник об этом не догадывается и атакует именно Вас. Поэтому на помощь со стороны напарника особенно не рассчитывайте. А вот использовать его как летающий склад с боеприпасами — милое дело. Как только Вы нажмете клавишу «N», Ваш напарник займет наиболее выгодную позицию для стрельбы ракетой по выбранной Вами цели. Подсветка осуществляется Вашим лазером, поэтому все правила использования Hellfire остаются в силе. Единственный недостаток состоит в том, что, как только Ваш напарник выдвигается вперед, весь огонь средств ПВО сосредоточивается на нем. Поэтому старайтесь не злоупотреблять доверием младшего товарища и не держите его под огнем дольше, чем это необходимо. Как только Вы переключитесь на другой вид оружия, Ваш напарник вернется в боевой порядок.

Немного о тактике

Несмотря на то, что окружающий мир представлен несколько упрощенно (два типа подвижных наземных объектов — танк Т-80 и ЗРК «ОСА» — и один тип воздушного противника — вертолет КА-50), для успешного выполнения заданий необходимо проявлять смекалку и обладать некоторыми навыками. Для формирования последних настоятельно рекомендуется, прежде чем приступать к боевым вылетам, пройти тренировочный курс. Кстати, успешно решившие все тренировочные задачи награждаются медалью. Правда, сам процесс награждения не идет ни в какое сравнение с F-117 или Wing Commander, но тут уж ничего не поделаешь! И учтите, что если Вы не выполнили любую из боевых задач, отмеченных красным цветом (приступать к ним разрешается только после завершения всех «зеленых» миссий), Вам придется вернуться к выполнению предыдущего задания.

Автору в результате исследований удалось установить одну из заповедей — не подпускай врага близко. Вертолет КА-50 обладает большей маневренностью и более мощным вооружением по сравнению с Comanche, и попытка «асовского» расстрела вертолета противника из пушек на ближних дистанциях приводит если не к поражению, то, по крайней мере, к повреждению вертолета. А это неприятно. К тому же, если Ваш вертолет получит в хвост очередь из пушки, велика вероятность, что у Вас выйдет из строя хвостовой ротор и Вы потеряете управление (отметим, что у КА-50 хвостового ротора нет вообще, и данный недостаток у этих вертолетов отсутствует).

Во-вторых, я настоятельно рекомендую пользоваться артиллерийской поддержкой. Некоторые задачи (по уничтожению крупных группировок противника) без нее не выполнить вообще.

И в-третьих, о напарнике. В первую очередь старайтесь использовать его боезапас. Обидно, если противнику удастся сбить загруженный вертолет, а у Вас осталось всего две-три ракеты!

Последнее замечание. RAH-66 при всех своих недостатках представляет собой отличный тренажер для любителей STEALTH операций. Последние задания требуют для их успешного выполнения тщательного использования складок местности. Вы должны суметь незаметно подкрасться к противнику, дать целеуказание средствам артиллерийской поддержки, так же незаметно отойти, а затем добить оставшихся!

Ну что же, любители «леталок», Вам взлет! ■

Boeing Sikorski RAH-66 Comanche
Тактико-технические характеристики

Двигатели	
Фирма-производитель	Аллисон-Гаррет
Модель	T800-LHT-801
Мощность	925 л.с.
Максимальная скорость	328 км/ч
Скороподъемность	360 метров/мин
Максимальная перегрузка	+3g
Перегоночная дальность полета	2335 км
Экипаж	2 человека
Габаритные размеры	
Длина фюзеляжа	13,22 м
Диаметр ротора	14,28 м
Ширина (габарит по шасси)	2,31 м
Ширина фюзеляжа	2,29 м
Высота (возвышение хвостового стабилизатора)	3,39 м
Вес	
Взлетный вес	7790 кг
Вес боевой нагрузки	4587 кг
Вес пустого вертолета	1185 кг
Емкость топливных баков	
Внутренние	984 л
На внешней подвеске	4542 л
Вооружение	
Ракетное вооружение	
Ракеты класса «воздух-земля» с лазерным наведением AGM-114 Hellfire	
Ракеты класса «воздух-воздух» с ИК головкой самонаведения AIM-92 Stinger	
20 мм автоматическое орудие	
Боекомплект	500 снарядов
Скорострельность	1500 выстрелов в минуту
Время наведения	около двух секунд
Навигационно-прицельная система	
Средства ночного видения на базе ПЗС-матриц второго поколения	
Ночной телевизионный прицел	
Лазерный измеритель дальности / прицел	
Интегрированная система сбора разведданных ATDS	
Встроенный в шлем пилота комбинированный прицел	
Основные классы боевых задач	
Тактическая разведка	
Сопровождение наземных/воздушных объектов	
Огневая поддержка в тактическом звене	
Борьба с вертолетами противника	
Наведение огня наземных средств дивизионного/корпусного звена	

Клавиши управления вертолетом

ESC	Меню управления программой
Левый индикатор/Правый индикатор	
F1/F7	Цифровая карта местности
F2/F8	Индикатор ракетных угроз
F3/F9	Телевизионный прицел
F4/F10	Ход выполнения боевой задачи
F5/F11	Повреждения
F6/F12	Зарезервировано для будущих версий имитатора
Обзор	
1	Кабина пилота
2	Левый борт
3	Правый борт
4	Оглянись!
5	Панорама
6	Внешняя камера
7	Сброс телекамеры
8	Вид от ранее сброшенной камеры
<	Увеличить масштаб цифровой карты
>	Уменьшить масштаб цифровой карты
Управление полетом	
E	Подняться выше («-» на дополнительной клавиатуре)
D	Опуститься ниже («+» на дополнительной клавиатуре)
Стрелки	Направление движения
*	Автоматическое огибание рельефа (дополнительная клавиатура)
Целеуказание	
TAB	Выбор цели
Enter	Выбор цели
Q	Выбор предыдущей цели на левый индикатор
W, S	Выбор следующей цели
Управление оружием	
Z	Автоматическая 20 мм пушка
X	Неуправляемые 70 мм ракеты
C	Ракеты с телевизионной системой наведения «Hellfire»
V	Ракеты с ИК самонаведением «Stinger»
B	Целеуказание наземным средствам
N	Целеуказание напарнику
M	Неуправляемые 70 мм ракеты (стрельба парами)
[	Предыдущий тип вооружения
]	Следующий тип вооружения
Пробел	Гашетка
Защита	
;	сброс уголкового отражателя (ракеты с РЛ головками наведения)
'	сброс ИК ловушки

Существуют ли шедевры, приносящие ежедневную пользу?
-Да, их поставляет компания
Радом

принтеры
EPSON

ЛАЗЕРНЫЕ

EPL-5200 (emul. HPIIIP, на 6000 копий)
HP LJ-4L (PCL-5, 300dpi, 26 fonts)

МАТРИЧНЫЕ

LX-100 (9-pin, 80 col, автоподача бумаги, трактор);
LQ-100 (24-pin, автоподача бумаги);
FX-1170 (9-pin, автоподача бумаги);
LQ-1070 (24-pin, 80 col, автоподача бумаги);
LQ-1170 (24-pin, 80 col, автоподача бумаги);

БАНКОВСКИЕ

DFX-8000 (18-pin, скорость печати - 1066 з/с
DFX-5000 (18-pin, скорость печати-533 з/с)

Оптовая и розничная торговля. У нас более 200 дилеров и 3 филиала. Приглашаем к сотрудничеству отраслевых и региональных дилеров.

ПРОГРАММНЫЕ ПРОДУКТЫ

Крупнейший дистрибьютор фирм SYMANTEC, BORLAND, MICROSOFT, LOTUS, NOVELL, COREL SYSTEM, SCO, AUTODESK, WORDPERFECT, COMPUTER ASSOCIATES, IBM, контролирующий более 50% продаж на рынке СНГ

ДИСКЕТЫ ФИРМЫ 3М

дискеты - 3,5; 5,25; картриджи к стриммерам - 120 мБ, 150 мБ, 250 мБ, 525 мБ,

БЫТОВАЯ ЭЛЕКТРОНИКА

фирм SONY, TOSHIBA, JVC, MOULINEX, ORION.

Екатеринбург
«Радом-Восток»,
Тел.:
(3432) 22-52-08.
Таганрог
«Радом-Т»,
Тел.:
(86344) 6-22-15.
Астрахань
Тел.:
(8510) 24-63-85

Торговые залы:
Москва,
Звенигородское шоссе д.9,
ул. Тверская, д.8,
ул. Новый Арбат, д.9.
Факс: (095) 259-2727,
Тел.: (095) 256-4473,
256-4030.
Телефон для оптовых покупателей:
(095) 256-1164.

Copyright © 1993 Радом® Н.А.А.

САМЫЙ НАДЕЖНЫЙ ПАРТНЕР

АО VIMCOM

INFA

Дилер CANON, HEWLETT PACKARD, NOVELL

Производственная фирма "ИНФА"

предлагает

- Компьютеры АТ любой конфигурации.
- Копировальные аппараты Canon.
- Телефаксы.
- Автоответчики.
- Различная оргтехника.
- Широкий выбор комплектующих к вычислительной технике: лазерные и матричные принтеры, стриммеры, винчестеры, модемы, контроллеры, дисководы, микросхемы, блоки питания, дискеты, расходные материалы.
- Установка ЛОКАЛЬНЫХ СЕТЕЙ любой конфигурации на всей территории России и других суверенных государств.
- Гарантийное и послегарантийное обслуживание.
- Программное обеспечение для банков: ОПЕРАЦИОННЫЙ ДЕНЬ БАНКА, "АСТРА".

Адреса торговых центров:

1. Москва, ул. Электродная, д. 10
(м. "Шоссе Энтузиастов")
Телефоны: (095) 176–0928, 176–1249, 306–6210, 306-6310.
Факс: (095) 176–7998, 306–3511.

2. Постоянно действующая выставка–продажа в павильоне "РАДИОЭЛЕКТРОНИКА" ВДНХ.
Телефон: (095) 181–9662.

3. Москва, ул. Балчуг, д. 22
(м. "Новокузнецкая").
Телефоны: (095) 233–5768, 231–8307.

4. Каширское ш., д. 16.
Телефон: (7 095) 112–4309.

Ваша программа, на Вашем компьютере, но гораздо быстрее!

Работу с этими и другими программами ускоряют сопроцессоры Intel

Математические сопроцессоры фирмы Intel:

— **привлекают крупнейших разработчиков программного обеспечения.** Свыше 600 таких компаний во всем мире ориентируются на математические сопроцессоры фирмы Intel, и уже свыше 2100 прикладных программ активно их используют;

— **предлагают проверенное решение для любого компьютера от XT до i486 SX.** Математические сопроцессоры фирмы Intel являются индустриальным стандартом. Во всем мире буквально миллионы пользователей с удовлетворением пожинают плоды повышенной производительности;

— **обеспечивают совместимость.** Поскольку сопроцессоры фирмы Intel разработаны специально для работы совместно с микропроцессорми Intel, на основе которых созданы все персональные компьютеры, они обеспечивают 100%–ную совместимость;

— **означают качество фирмы Intel.** Математические сопроцессоры, изготовленные фирмой Intel, являются наивысшим стандартом в мире и имеют пятилетнюю гарантию;

— **легко устанавливаются.** Надо лишь вставить сопроцессор в отведенное для него гнездо – и готово!

intel®